

Editorial Delta, S.A.

MI COMPUTER CURSO PRACTICO
DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

Publicado por Editorial Delta, S.A., Barcelona

Volumen VI-Fascículo 63

Director: José Mas Godayol
Director editorial: Gerardo Romero
Jefe de redacción: Pablo Parra
Coordinación editorial: Jaime Mardones
Francisco Martín
Asesor técnico: Ramón Cervelló

Redactores y colaboradores: G. Jefferson, R. Ford, F. Martín, S. Tarditti, A. Cuevas, F. Blasco
Para la edición inglesa: R. Pawson (editor), D. Tebbutt (consultant editor), C. Cooper (executive editor), D. Whelan (art editor), Bunch Partworks Ltd. (proyecto y realización)

Realización gráfica: Luis F. Balaguer

Redacción y administración:
Paseo de Gracia, 88, 5.°, 08008 Barcelona
Tels. (93) 215 10 32 / (93) 215 10 50 - Télex 97848 EDLTE

MI COMPUTER, *Curso práctico del ordenador personal, el micro y el miniordenador*, se publica en forma de 96 fascículos de aparición semanal, encuadernables en ocho volúmenes. Cada fascículo consta de 20 páginas interiores y sus correspondientes cubiertas. Con el fascículo que completa cada uno de los volúmenes, se ponen a la venta las tapas para su encuadernación.
El editor se reserva el derecho de modificar el precio de venta del fascículo en el transcurso de la obra, si las circunstancias del mercado así lo exigieran.

© 1983 Orbis Publishing Ltd., London
© 1984 Editorial Delta, S.A., Barcelona
ISBN: 84-85822-83-8 (fascículo) 84-7598-034-7 (tomo 6)
84-85822-82-X (obra completa)
Depósito Legal: B. 52/1984

Fotocomposición: Tecfa, S.A., Pedro IV, 160, Barcelona-5
Impresión: Cayfosa, Santa Perpètua de Mogoda (Barcelona) 278503
Impreso en España-Printed in Spain-Marzo 1985

Editorial Delta, S.A., garantiza la publicación de todos los fascículos que componen esta obra.

Distribuye para España: Marco Ibérica, Distribución de Ediciones, S.A., Carretera de Irún, km 13,350. Variante de Fuencarral, 28034 Madrid.
Distribuye para Colombia: Distribuidoras Unidas, Ltda., Transversal 93, n.° 52-03, Bogotá D.E.
Distribuye para México: Distribuidora Intermex, S.A., Lucio blanco, n.° 435, Col. San Juan Tlihuaca, Azcapotzalco, 02400, México D.F.
Distribuye para Venezuela: Distribuidora Continental, S.A., Edificio Bloque Dearmas, final Avda. San Martín con final Avda. La Paz, Caracas 1010.

Pida a su proveedor habitual que le reserve un ejemplar de MI COMPUTER. Comprando su fascículo todas las semanas y en el mismo quiosco o librería, Vd. conseguirá un servicio más rápido, pues nos permite realizar la distribución a los puntos de venta con la mayor precisión.

Servicio de suscripciones y atrasados (sólo para España)

Las condiciones de suscripción a la obra completa (96 fascículos más las tapas, guardas y transferibles para la confección de los 8 volúmenes) son las siguientes:

a) Un pago único anticipado de 19 425 ptas. o bien 8 pagos trimestrales anticipados y consecutivos de 2 429 ptas. (sin gastos de envío).

b) Los pagos pueden hacerse efectivos mediante ingreso en la cuenta 6.850.277 de la Caja Postal de Ahorros y remitiendo a continuación el resguardo o su fotocopia a Editorial Delta, S.A. (Paseo de Gracia, 88, 5.°, 08008 Barcelona), o también con talón bancario remitido a la misma dirección.

c) Se realizará un envío cada 12 semanas, compuesto de 12 fascículos y las tapas para encuadernarlos.

Los fascículos atrasados pueden adquirirse en el quiosco o librería habitual. También pueden recibirse por correo, con incremento del coste de envío, haciendo llegar su importe a Editorial Delta, S.A., en la forma establecida en el apartado b).

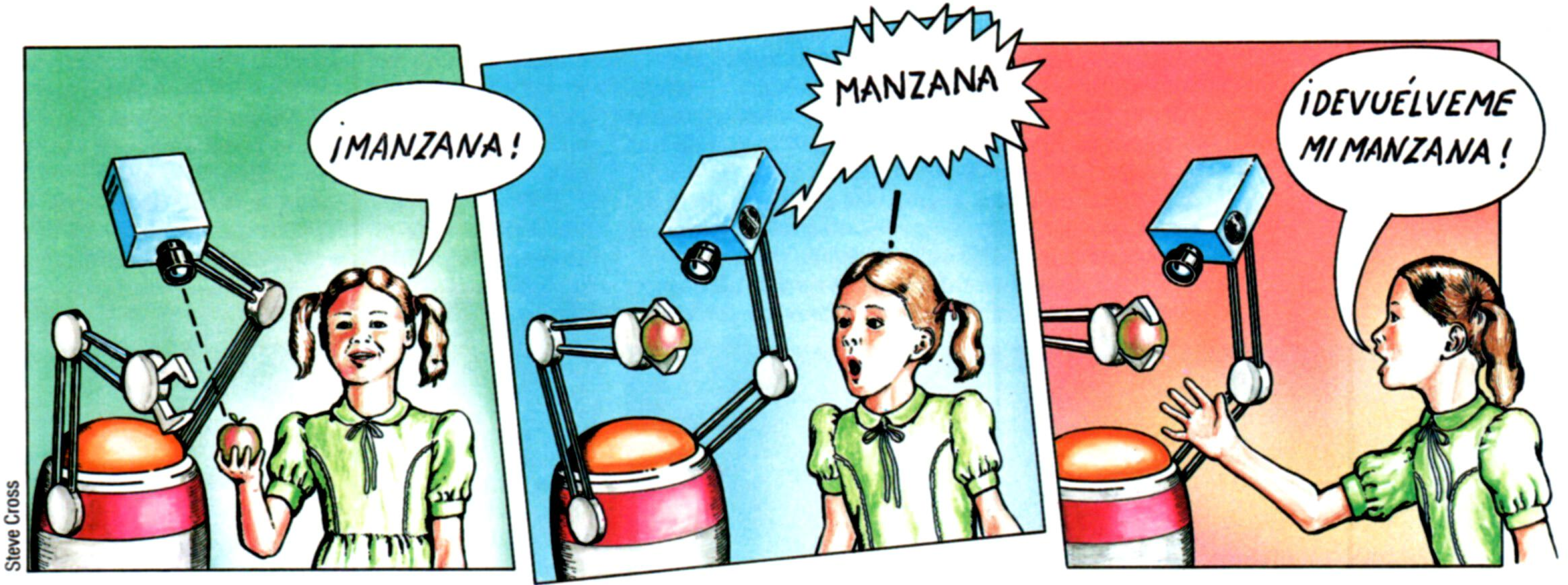
Para cualquier aclaración, telefonear al (93) 215 75 21.

No se efectúan envíos contra reembolso.





Hablemos del habla



Ver, comparar y hablar
Cuando una persona ve un objeto —una manzana, por ejemplo— y le da un nombre, existe una comprensión del significado de "manzana". El robot puede reconocer visualmente el objeto mediante la comparación de lo que ve con una imagen interna, y puede repetir el patrón de sonido que tiene almacenado para acompañar a "manzana". Pero no comprende en absoluto que el objeto es una fruta comestible ni, y quizá sea más importante, que la manzana en realidad le "pertenece" a la persona. Esto, por supuesto, es algo que el humano entiende perfectamente

Para entender la dificultad de hacer hablar a un robot es necesario considerar algunas teorías acerca de la adquisición del lenguaje

El estudio del habla humana ha propiciado la existencia de dos escuelas de pensamiento: la de quienes piensan que las aptitudes del lenguaje son innatas (algo con lo que ya hemos nacido) y la de quienes opinan que el lenguaje se adquiere o se aprende. Los psicólogos que sostienen que éste es innato señalan que el hombre es la única criatura que se comunica mediante el lenguaje. Quienes piensan que es adquirido aluden a experimentos realizados con animales a los cuales se les ha enseñado a comunicarse con los humanos a través de signos.

Si las personas aprendieran a hablar simplemente por hallarse inmersas en un mundo que se relaciona por medio del habla, entonces sería lógico buscar algún método para conseguir que los robots hicieran lo mismo. Al fin y al cabo, la vida sería muchísimo más sencilla si un robot pudiera aprender el lenguaje por el mero hecho de escuchar cómo lo hablan quienes lo rodean.

Se han llevado a cabo ciertos intentos para lograr que un ordenador amplíe sus conocimientos de gramática gracias a que se le proporcionan ejemplos extras de estructuras gramaticales de frases, mientras que con otros experimentos se ha intentado conseguir que un robot aprenda palabras y morfemas (elementos del lenguaje) nuevos en cualquier idioma simplemente mostrándoselos. Pero aún no se ha desarrollado ningún sistema que haya logrado enseñarle a un robot a aprender el habla.

En consecuencia, en la práctica, las aptitudes del robot para adquirir un lenguaje se basan en el supuesto de que éste es innato, que las aptitudes no se aprenden y que lo que debemos hacer es elaborar las reglas del lenguaje y fijárselas al robot permanentemente como si él hubiera nacido con ellas. En general, esto consta de dos fases diferenciadas: el análisis sintáctico y el análisis semántico.

El *análisis sintáctico* trata de la gramática de lo que se está diciendo y decodifica la estructura superficial del mensaje o codifica el mensaje en una forma gramatical correcta para que el robot la transmita. El método más común para realizar esto es mediante un *árbol de análisis* (*parsing tree*), que gradualmente va descomponiendo, o construyendo, una oración a partir de las diversas partes de la misma. Ésta no es una tarea sencilla; no obstante, poco a poco se está abordando con cierto éxito.

El *análisis semántico* es mucho más complicado e implica elaborar el sentido del mensaje (cuando el robot está escuchando hablar a alguien) o elaborar cuál es el mensaje que se necesita comunicar (cuando el robot desea hablarle a uno). El problema del análisis semántico es que el lenguaje no es independiente de su entorno: su significado depende del contexto en el que está incluido (y esto no se aplica sólo al entorno hablado, sino al contexto global del mensaje). Este contexto puede abarcar el conocimiento del mundo circundante mientras uno está hablando, así como el conocimiento que cada una de las partes posee acerca de la otra.

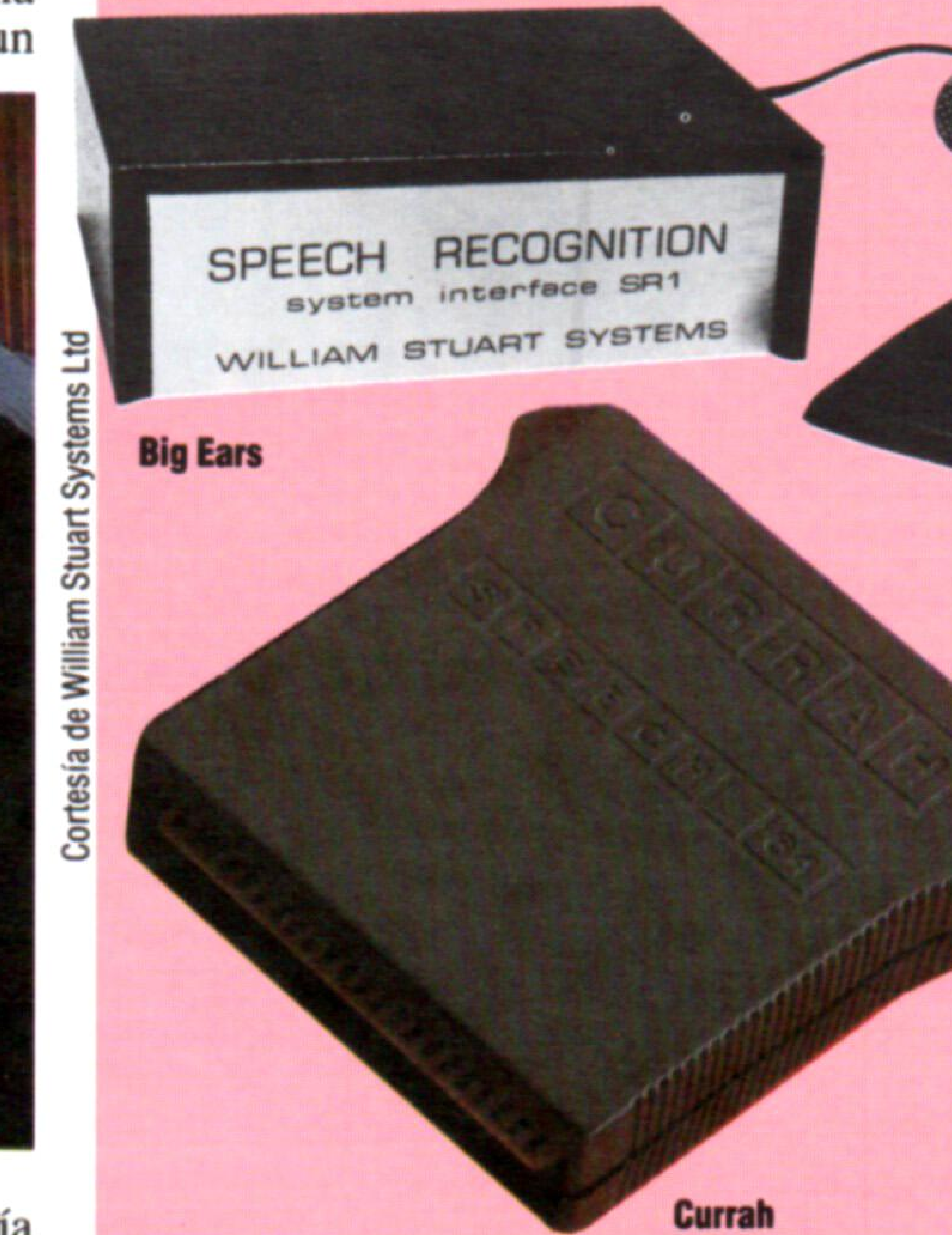
Este enfoque se ha adoptado en experimentos dirigidos por el científico informático Terry Winograd, quien escribió un programa que permitió que un robot entendiera lo que se le decía y actuara a tenor de las instrucciones. Sin embargo, Winograd utilizó una simulación por ordenador de un robot que sólo era capaz de operar en un entorno definido muy minuciosamente. En este caso, el mundo del robot consistía en una cantidad de bloques de construcción que era capaz de manipular. El pro-

grama de Winograd, conocido como SHRDLU, fue capaz de efectuar un buen análisis semántico, pero el entorno que era capaz de discernir era extremadamente simple. Un robot que hubiera estado operando en el caos que es el mundo real hubiese tenido muchísimas más dificultades para entender lo que se le estaba diciendo.

Con el objeto de que los robots empleen provechosamente el lenguaje, ha de haber un mensaje que se transmita de la persona al robot, y viceversa. Para los autómatas cibernéticos, hablar es relativamente fácil, porque, dado que sus conocimientos son tan restringidos, es probable que lo que deseen expresar sea muy limitado. Para ellos es mucho más difícil entender lo que una persona pueda decirles, porque todo lo que se desee comunicarles será mucho más difícil de analizar.

En una época se pensó que la entrada en forma de voz a los robots se analizaría llevando a cabo un análisis sintáctico de la entrada y que éste revelaría el significado del mensaje. Pero trabajos recientes han puesto de relieve la importancia del conocimiento acerca del mundo circundante y del contexto en el cual se expresa el mensaje. Esto ha llevado a experimentos en los cuales se realiza un análisis sintáctico tentativo de la señal del habla para formular una primera conjetura sobre el significado. Luego, a la luz de lo que el robot sabe acerca del mundo y de las cosas que es probable que se digan en su entorno, el robot revisa su análisis sintáctico original con la finalidad de ir acercándose gradualmente a un análisis correcto de lo que se está diciendo. No obstante, esto dista mucho de lo que hoy puede hacer cualquier robot. Ahora vamos a estudiar cómo hablan y cómo entienden la voz los sistemas actuales de robots.

Apricot F1



Cortesía de William Stuart Systems Ltd

Oír y hablar
Crear habla por ordenador y habla para robots es bastante fácil. Existen a la venta dispositivos para síntesis de voz, como el Currah, incluso para los ordenadores personales más pequeños. Pero el reconocimiento es más difícil debido a las variaciones de la forma en que los humanos pronuncian los sonidos vocálicos y a causa de la cantidad de memoria y de potencia de proceso que requieren para manejar un vocabulario de más de unas pocas palabras. Sistemas como el Big Ears (literalmente "grandes orejas") y el Apricot F1 llevan incorporada una pequeña gama de instrucciones reconocibles, pero demasiado pocas como para cubrir más de una cantidad mínima de operaciones

Síntesis de voz

El método más simple de síntesis de voz emplea un magnetófono en el cual hay un mensaje pronunciado por un ser humano que está grabado en cinta y que el robot reproduce en un momento adecuado. Puede que esto no se parezca mucho a la idea que se formó usted cuando pensó por primera vez en el habla del robot, pero es el punto de partida de todos los sistemas de síntesis de voz. En primer lugar, analizaremos las limitaciones de este método y luego veremos cómo superarlas.

La limitación más evidente es que un magnetófono es mecánico, caro, voluminoso y susceptible de romperse. De modo que el siguiente paso consiste en tomar el mismo mensaje y convertirlo a una forma digital, de modo que pueda ser almacenado en un chip de la memoria del robot. Ello se realiza mediante un convertidor de analógico a digital, en el cual se utilizan números para representar la forma de onda de la voz, que varía de manera constante. Éste es el mismo procedimiento que se aplica para la grabación digital de música en, por ejemplo, los sistemas de disco compacto.

Este método tiene, asimismo, sus inconvenientes. Uno de los problemas fundamentales es que una señal digitalizada ocupa muchísimo lugar en la memoria. La grabación en disco compacto muestrea la señal acústica alrededor de 44 000 veces por segundo con una resolución de aproximadamente 16 bits (es decir, la amplitud de la forma de onda en cualquier momento se almacena como un número de 16 bits, que permite discernir 2^{16} niveles, donde $2^{16} = 65\,536$). Utilizando este sistema, cada segundo de la grabación ocuparía 88 000 bytes. Evidentemente, un mensaje hablado supera la capacidad de almacenamiento de cualquier microordenador. Sin embargo, esta velocidad de muestreo sólo es aplicable a la reproducción de sonido de alta fidelidad; se podría construir un sistema de habla sencillo con una resolución de ocho bits y una velocidad de muestreo de 3 000 muestras por segundo, ¡lo que sólo ocuparía tres Kbytes!

Sin embargo, con el fin de dejar libre la máxima cantidad de espacio de memoria, es necesario realizar más economías. Los lingüistas han descubierto que el lenguaje hablado se puede descomponer convenientemente en unidades que se denominan fonemas. En conjunto, la opinión general coincide en el hecho de que existen alrededor de 40 fonemas diferentes para la mayoría de los lenguajes hablados, de modo que es posible almacenar la exacta información acústica necesaria para describir cada



uno de estos 40 fonemas y luego utilizar éstos como la base del habla del robot. Típicamente, la información relativa a los fonemas está retenida en un chip sintetizador de voz que se fabrica comercialmente y todo lo que el robot ha de hacer es encadenar estos fonemas para generar el mensaje requerido. Este mensaje se suele almacenar como una serie de números en la memoria del ordenador.

La mayoría de los sintetizadores de voz que existen en uso se pueden programar escribiendo el mensaje que el robot ha de expresar en una versión fonética del inglés. Por consiguiente, el mensaje "*Can you come here?*" (¿Puedes venir aquí?) se escribiría "kan yew kum heah", y esto sería suficiente para que el chip sintetizador produjera la serie correcta de sonidos. Esta notación no es exactamente la misma que utilizan los lingüistas para describir los fonemas (ellos poseen su propio alfabeto especializado) pero es suficiente para los robots.

Llegados a este punto, observará que el robot ya no está empleando un mensaje pregrabado: está realmente generando mensajes propios. Por esta razón se puede lograr que el robot diga lo que deseemos sin necesidad de tener el mensaje entero grabado con anterioridad.

Por lo tanto, si así nos lo propusiéramos, podríamos tratar de programar algunas de las reglas de la gramática en un intento de hacer que el robot dijera cosas originales. Pero, como ya hemos mencionado, el número de cosas diferentes que podría expresar un robot es más bien limitado, de manera que no existe ninguna necesidad de una complejidad excesiva, a menos que nos dejemos llevar por un espíritu aventurero o por la curiosidad de ver qué es lo que se puede hacer.

Si ha escuchado alguna vez un sintetizador de voz en un robot, sabrá que la calidad de la voz, si bien por lo general es comprensible, de ningún modo es perfecta. Ello se debe a dos factores. El primero es que la forma que toma un fonema cuando lo utiliza un humano varía considerablemente en virtud de los fonemas que lo preceden y le siguen. El segundo factor es que el sonido de la voz humana varía según el significado que deseemos transmitir. "¿Quieres sentarte?" y "¿Quieres *sentarte?*" son dos mensajes escritos de manera idéntica, pero sonarán muy distintos si el primero lo dice un cortés anfitrión a su huésped y el segundo lo pronuncia un maestro exasperado. Se han realizado algunos intentos por captar esta entonación en los sistemas de síntesis de voz, pero su aplicación es difícil, dado que un robot no conoce el significado de las palabras que está pronunciando.

Reconocimiento de voz

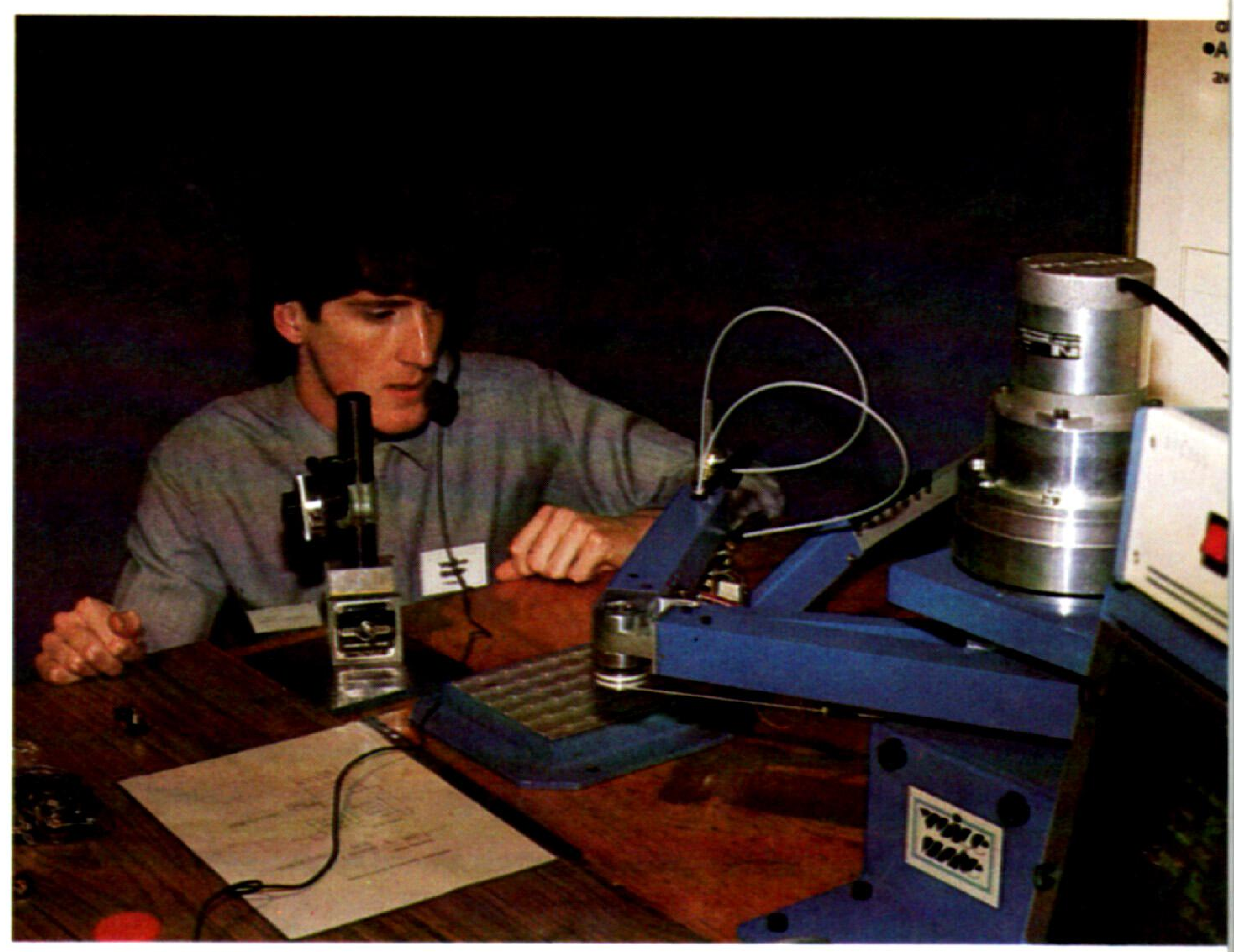
El problema inevitable a resolver cuando se desarrolla un sistema de reconocimiento de voz es que las cosas que desearíamos decirle a un robot, y las formas de expresarlas, son muchas y muy diversas. Un enfoque posible a este problema sería utilizar una grabación en cinta de todo lo que quisiéramos que el robot comprendiera. Cuando habláramos, él podría entonces explorar todas sus grabaciones en cinta y buscar la que más se pareciera al mensaje que acabara de escuchar; y así es, en principio, cómo reconocen la voz muchos robots. Almacenan "modelos" internos de mensajes hablados y, cuando se les habla, simplemente buscan el modelo que ofrece la mayor similitud. Estos modelos generalmente se obtienen entrenando al robot (repitiendo varias veces una palabra o frase) hasta que adquiera un modelo "promedio" de lo que hemos dicho. Este método funciona bien si sólo se tiene una pequeña cantidad de cosas que decirle al robot y si vamos a decírselas siempre más o menos de la misma forma. Se utiliza para aquellos robots que responden a instrucciones sencillas, tales como "adelante", "gira a la izquierda", etc.

Sin embargo, éste es un problema comparativamente simple que se conoce como *reconocimiento de voz discreta*, porque cada ítem hablado es "discreto", es decir, está separado de otros mensajes mediante una breve pausa durante la cual no se dice nada.

El verdadero problema surge cuando deseamos hablarle al robot empleando *voz continua*, que es la que utilizamos normalmente. Pruebe decir "Es un hermoso día de verano", y escuche atentamente lo que ha pronunciado. Descubrirá que se oye algo así como "Esunher mosodía deverano", con las palabras y los sonidos chocando unos contra otros.

La forma en la cual una persona aborda este problema cuando está escuchando hablar a otra consiste en adivinar qué es lo que el interlocutor intenta decir (lo que no suele ser difícil) y utilizar esta conjetura para decodificar el mensaje. Pero para que un robot hiciera eso, habría que saber muchísimo acerca de qué es probable que se diga y quiera decirse con ello: una tarea muy complicada.

En general, el empleo de la síntesis de voz en robots se está convirtiendo en algo común, si bien aún quedan cosas por hacer en cuanto a elevar la calidad de su habla. El reconocimiento de voz es una tarea mucho más difícil y, en la actualidad, lo más que se puede lograr con cierta facilidad es dotar al robot de una comprensión del habla equivalente a la de un perro entrenado que responda a las instrucciones habladas, siempre y cuando éstas no sean muy numerosas. No obstante, existe un extraordinario interés por resolver todos los problemas que entraña el habla del robot.



Óyeme, siénteme
El Voicemate es un brazo-robot controlado por voz que desarrolló el departamento de ingeniería del Politécnico de Newcastle (Gran Bretaña) para uso industrial y de laboratorio

Cortesía de Newcastle Polytechnic

Campo de fútbol

En esta ocasión examinaremos el paquete "Multiplan", una completa hoja de trabajo electrónica para el Commodore 64

En *Multiplan*, primer programa de hoja electrónica producido por Microsoft, se han incorporado muchas ideas avanzadas, desarrolladas a partir de paquetes de hoja electrónica anteriores. Éstas incluyen la capacidad de actuar sobre grupos de celdas que se describen por nombre; clasificar un grupo de entradas de acuerdo a un criterio especificado; dividir pantallas, permitiendo visualizar simultáneamente zonas diferentes de la hoja electrónica; buscar rápidamente en la información retenida en forma de tabla y luego producir un valor requerido, y llevar a cabo construcciones condicionadas IF...THEN, entre otras. Disponible originalmente sólo para ordenadores de alto nivel tales como el IBM PC y el Apple II, recientemente ha salido al mercado el *Multiplan* para el Commodore 64.

El modelo que construiremos con este programa simplifica la tarea de mantener al día las estadísticas. Los datos que utilizamos se refieren al fútbol americano, pero la estructura se puede adaptar para otros deportes.

Después de cargar el *Multiplan*, se visualiza una hoja de trabajo de formato estándar de 63 columnas y 255 filas. Tanto las filas como las columnas están numeradas, de modo que la celda base, en la esquina superior izquierda, es la celda R1C1 (por Fila (Row) 1 Columna 1). En la parte inferior de la pantalla aparece un menú de instrucciones, con un cursor que destaca la primera opción, Alpha. Las instrucciones del menú de *Multiplan* se pueden seleccionar pulsando la barra espaciadora para desplazar el cursor hasta la instrucción deseada y pulsando Return, o tecleando la primera letra de ésta.

Con frecuencia, la selección de una instrucción hace que se visualice un submenú que ofrece una amplia variedad de opciones para formateo de datos, administración de memoria, etc. Pulsando una tecla de letra se accede a una instrucción, de modo que uno debe digitar A de Alpha antes de entrar texto. Los números se pueden entrar directamente, pero las fórmulas deben ir precedidas por un signo más (+) o igual (=).

Las dos primeras filas de la hoja de trabajo retienen títulos. Por conveniencia, hemos formateado las celdas desde R1C1 hasta R2C5 para texto continuo, lo que permite extender el texto más allá de los límites de la celda. Ello se consigue digitando:

F(formato) C(eldas) R1C1:R2C5

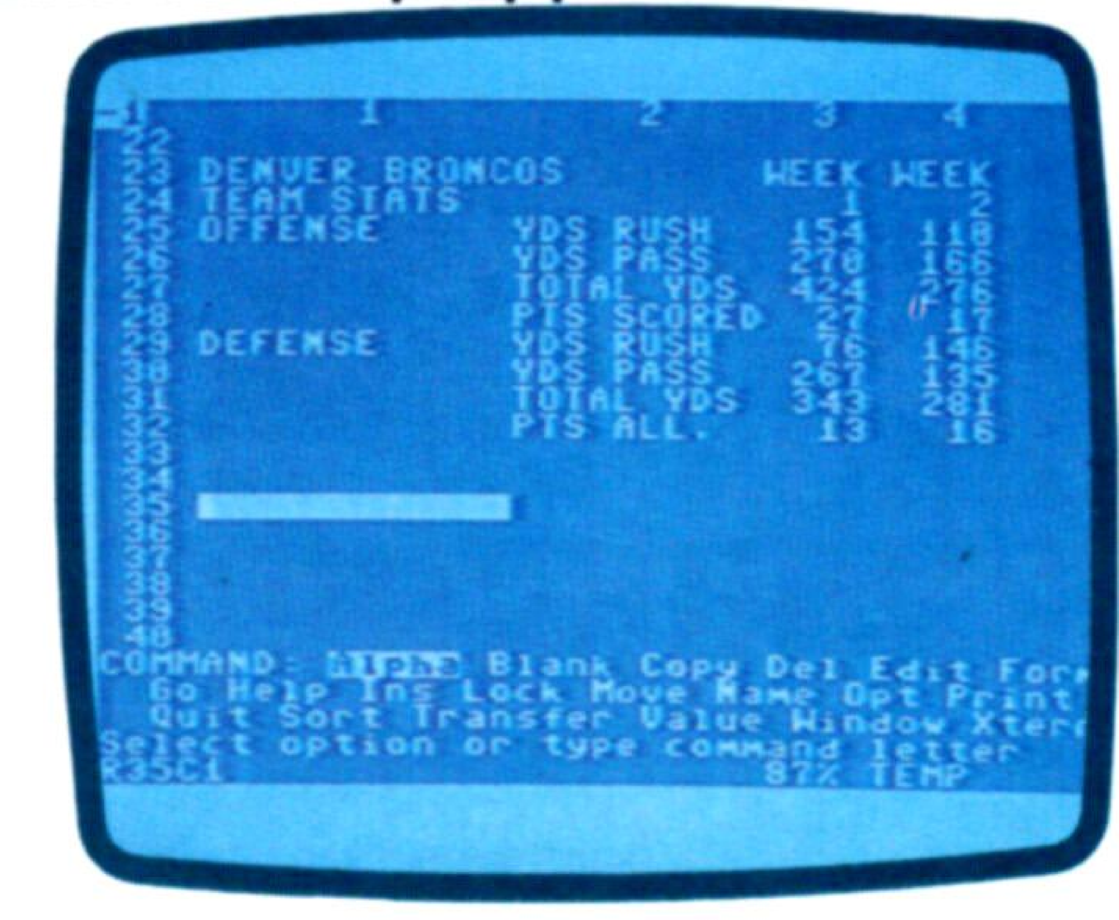
colocando luego el cursor encima de la palabra Cont y pulsando Return. Los dos puntos se utilizan para indicar una serie de celdas. Algunas columnas han variado su ancho para dar cabida a sus entradas.

La hoja de trabajo tiene dos porciones principales: una retiene información para un equipo específico durante un período de nueve semanas, y la otra es una tabla de los resultados gana/pierde para todos los equipos de la misma "conferencia" (véase texto sobre fútbol americano en el margen de la página contigua). Una vez construido el esqueleto del modelo, gran parte de los datos semanales habrán de entrarse a mano, con unas pocas fórmulas para mantener actualizados los totales a medida que avanza la temporada.

Resultados de la Liga



Informe del rendimiento por equipos



Ian McKinnell

La primera porción de la hoja de trabajo es una tabla. Los totales de ésta se deben actualizar cada semana, después de que hayan jugado los equipos. Mediante el almacenamiento de la información en una tabla podemos sacar partido de una de las características de *Multiplan*: la facilidad SORT (clasificar). Hemos entrado los nombres, categorías y datos de los equipos. El orden inicial de éstos está basado en las posiciones que ocupan en la liga. Sin embargo, la tabla se puede clasificar por cualquiera de las categorías almacenadas. *Multiplan* clasificará una serie especificada de filas de una columna por orden numérico ascendente o descendente. El texto se clasifica alfabéticamente.

Como ejemplo de función SORT, vamos a reacomodar los datos reflejados por los nombres de los equipos en orden alfabético. Después de digitar S, de SORT, *Multiplan* visualiza lo siguiente:

SORT by column:__between rows:__and:__
order: >< (CLASIFICAR la columna__entre las filas__ y__por orden ><)

Nosotros queremos clasificar la columna 1 entre las filas 7 y 21 en orden ascendente (>). La pulsación de Return hace que *Multiplan* reorganice los nombres alfabéticamente y reacomode los datos en consecuencia. Por ejemplo, todos los números relativos a Miami se desplazarán junto con Miami hasta su nueva posición. Simplemente cambiando la columna clave de SORT podemos reacomodar la tabla en función del equipo con mayor puntuación, los equipos que han permitido que sus oponentes marquen la menor cantidad de puntos, etc.

Tabla clasificada



Ahora desplazamos los datos de la pantalla pulsando la flecha del cursor que señala hacia abajo y hallamos la segunda porción de la hoja de trabajo: el informe del rendimiento individual por equipos. Aquí se utilizarán dos fórmulas. La primera es una simple fórmula SUM (suma), para llevar un total actualizado de los valores semanales. Localice la columna etiquetada TOTALS (totales) de la sección dos (R24C12). Desearemos que *Multiplan* sume los valores de cada columna semanal. Dado que querremos copiar la fórmula de modo que se hallen los totales para todas nuestras categorías (cubriendo la zona desde R25C12 hasta R32C12), necesitamos incorporar una referencia a una celda relativa. En *Multiplan* esto se realiza señalando las celdas activas con el cursor. La fórmula se entra digitando: =SUM(y pulsando luego la tecla de la flecha hacia la izquierda hasta que el cursor quede situado en R25C3. Entonces digitamos el carácter dos puntos para indicar que se está especificando una serie de celdas. El cursor regresa automáticamente hasta la celda en la cual uno está entrando la fórmula, de modo que pulse una sola vez la flecha hacia la izquierda, con el cursor situado en R25C11, y después pulse Return. La fórmula se verá así:

=SUM(R [-9]C:R[-1]C)

y se visualizarán los totales para los valores retenidos en la gama descrita. Ahora copie la fórmula en la serie de celdas desde R26C12 hasta R32C12 manteniendo el cursor en la fórmula y empleando la instrucción COPY: C(opiar) d(own) (abajo) 7 filas. Utilice el mismo proceso para hallar los totales para YDS RUSH e YDS PASS. La fórmula SUM se coloca en la celda R27C3, para las yardas ganadas en ataque, y en R31C3, para las yardas cedidas al equipo rival. La fórmula se copia en las ocho columnas de la derecha, para cubrir el período completo de nueve semanas.

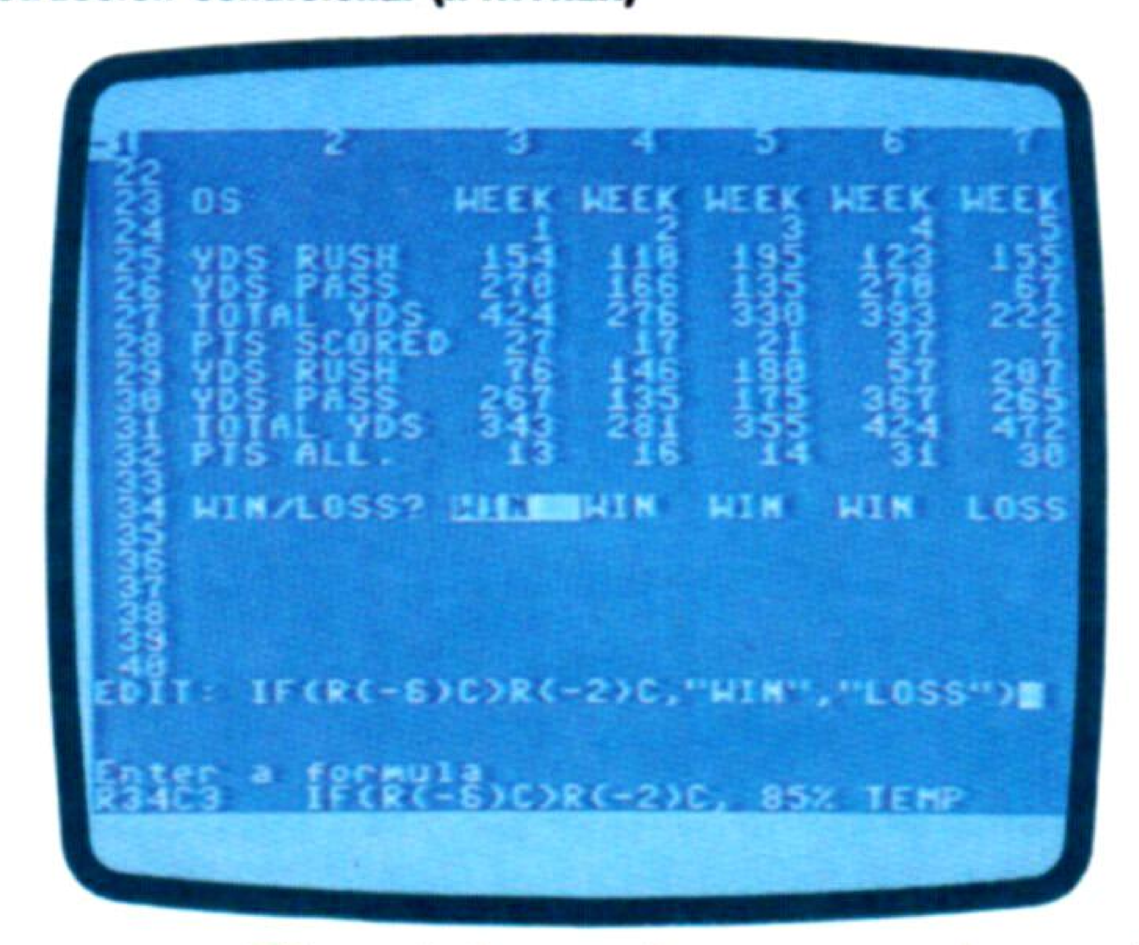
La segunda fórmula, empleando la sentencia IF, es un poco más complicada pero sumamente útil. En nuestro modelo dejaremos que Multiplan determine si un partido se ha ganado o se ha perdido mediante la comparación de los puntos totales de dos categorías: Points Scored ("puntos ganados" por nuestro equipo) y Points Allowed ("puntos cedidos": el marcador del rival). Necesitamos una sentencia como ésta: If Puntos Ganados > Puntos Cedidos, print WIN (ganado), else print LOSS (perdido).

Nuevamente, queremos referencias relativas y, por lo tanto, utilizamos movimientos del cursor para indicar las posiciones de los dos valores. Coloque el cursor en R34C3, etiquetada WIN/LOSS, y entre la fórmula:

IF(R[−6]C>R[−2]C,"WIN","LOSS")

Observe que el texto que se utiliza dentro de las fórmulas debe ir encerrado entre dobles comillas y que se necesitan paréntesis para encerrar las condiciones. Ahora copie la fórmula a lo largo de la fila,

Construcción condicional (IF...THEN)

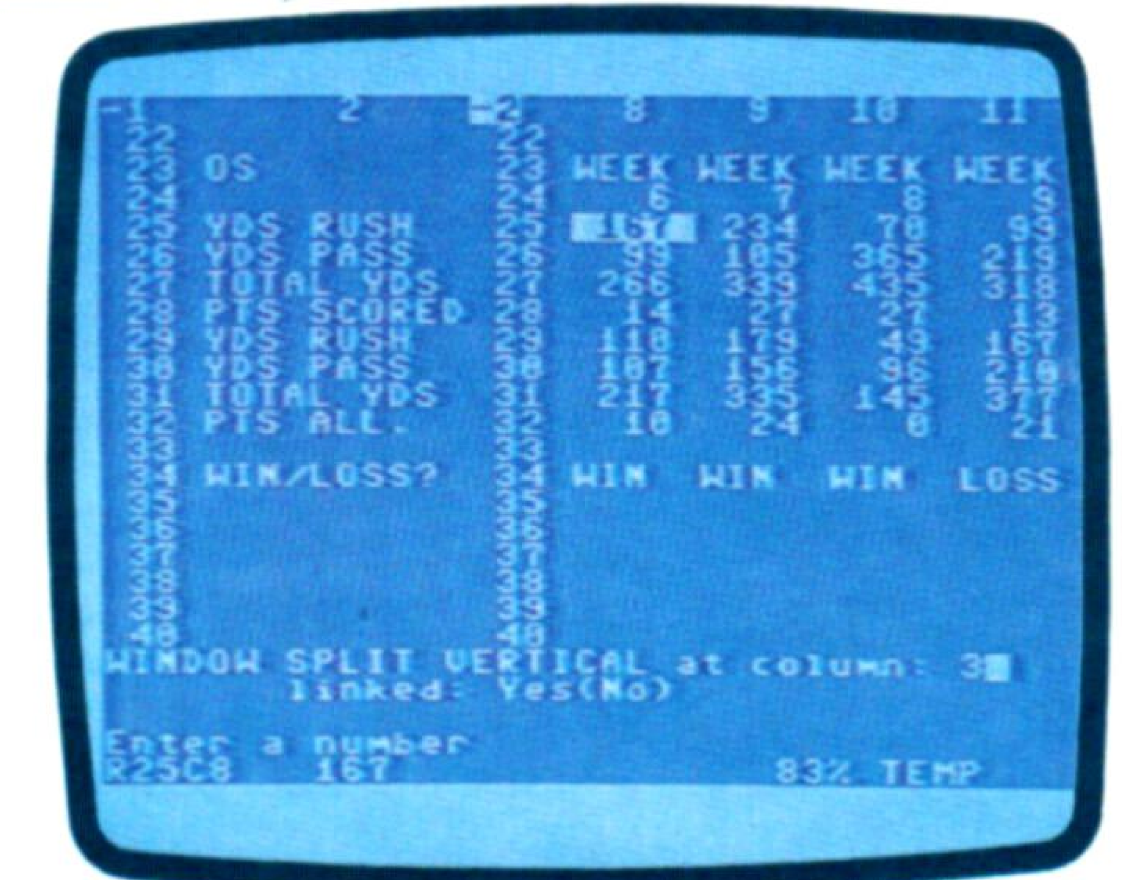


como antes. El modelo que hemos creado contiene datos para nueve semanas. Debido a las dimensiones de la pantalla, no podemos ver ni las etiquetas del margen izquierdo de nuestro informe por equipos, ni los totales de la derecha. Pero podemos dividir la pantalla en dos ventanas, que se puedan desplazar juntas o por separado.

Ahora desearemos dividir la pantalla verticalmente en la columna 3, de modo que pulse W(indow) (ventana) y S(plit) (dividir), seguidas de V(ertical). Entonces el programa visualizará:

WINDOW SPLIT VERTICAL at column:__linked YES/NO. (División vertical de ventana por la columna__unidas SI/NO)

Visualización con pantalla dividida



Usted tendrá que entrar la columna 3, desplazar el cursor hasta NO y pulsar Return. Si las ventanas no están unidas se les puede desplazar por separado. Ahora las etiquetas se pueden ver independientemente de qué porción de la hoja de trabajo se esté visualizando. Para cerrar la ventana digite W(indow) (ventana) C(lose) (cerrar), seguidas de su número.



Kevin Jones

Fútbol americano
Para quienes no estén familiarizados con el fútbol americano, he aquí una breve explicación de los términos que se utilizan en él. Dos equipos compuestos por 11 jugadores se turnan para tratar de desplazar un balón ovoide a través de una línea de gol. Las metas opuestas están separadas por 100 yardas (91 m). El balón lo puede transportar un corredor, arrojándolo hacia adelante como un pase, o pateándolo por entre los postes de la portería. Se consiguen tres puntos por un *kick*: patada (llamado *field goal*: tiro libre); se otorgan seis puntos por un *carry* (denominado *touchdown*: tanto marcado al tocar el suelo con el balón detrás de la meta del adversario), y uno por un *kick* después de un *touchdown* (lo que se conoce como un punto tras *touchdown*).
Cada equipo dispone de cuatro intentos o *downs* para llevar el balón 10 yardas hacia la meta contraria. Si tiene éxito, puede seguir hacia la meta con otros cuatro *downs*. Cuando un jugador lleva el balón, ello se denomina un *rush* (carrera), y el número de yardas que recorra un equipo constituye un índice de cuán lejos se ha llevado el balón durante el partido. El número de yardas *passing* señala la distancia a la cual se ha lanzado el balón.
En la National Football League (la principal liga profesional de Estados Unidos) hay dos "conferencias": la American Conference y la National Conference. Los equipos juegan una temporada de 16 encuentros que comienza en septiembre y termina en enero con la *Super Bowl* (Supertazón: por la forma del típico estadio de fútbol). En ésta participa el mejor equipo de cada conferencia

Aventura y emoción

Iniciamos un gran proyecto en que guiaremos al lector a través de las etapas de creación de un juego de aventuras

Los juegos de aventuras se hicieron populares a principios de los años setenta, cuando se ideó el juego *Calabozos y dragones* (*Dungeons and dragons*). En este juego los participantes asumen el papel de varios personajes de un mundo imaginario diseñado por el "amo de la mazmorra" (*dungeon master*). Este mundo imaginado se suele componer de un intrincado laberinto de salas que contienen objetos y peligros que han de ser superados por los jugadores. Generalmente, el objetivo del juego no es otro que escapar del laberinto, en la mayoría de los casos rescatando por el camino a alguien o algo. Los programadores de ordenadores centrales fueron los primeros que aplicaron el juego a éstos, construyendo complejos laberintos para que deambularan por ellos otros usuarios de ordenadores centrales. La ventaja del *Calabozos y dragones* basado en ordenador era que el amo de la mazmorra y los jugadores no tenían necesariamente que estar presentes al mismo tiempo, permitiendo que las personas jugaran en cualquier momento que lo desearan. Desde entonces, el juego del tipo *Calabozos y dragones* ha ensanchado considerablemente tanto su horizonte como su atractivo: el MUD (*Multi-User Dungeon*: mazmorra multiusuario; véase p. 864) es un buen ejemplo del grado de refinamiento que han alcanzado algunos de ellos.

Algunos juegos de aventuras están basados exclusivamente en texto, mientras que otros se valen del color y los gráficos para proporcionar imágenes en pantalla. Sin embargo, algunos críticos sostienen que la adición de gráficos supone la ocupación de un valioso espacio de memoria que, de lo contrario, se podría utilizar para agregar más enredos a la estructura del juego. Asimismo, señalan que la imagen gráfica por ordenador de una escena o de un lugar no tiene comparación posible con la propia imaginación de uno al evocar una imagen basada

El mapa

El primer paso en el diseño de un juego de aventuras consiste en trazar un mapa que incluya los diversos escenarios que pueden visitar los jugadores. Cada uno de los escenarios lleva una breve descripción de la escena, indicando si hay presente algún objeto y si el escenario posee algún significado especial en el contexto del juego. La numeración permite codificar y almacenar el mapa en el programa

POSIBLES
MOVIMIENTOS
DESDE EL
ESCENARIO 7
SL$(7)="08000306"

08
06 SEND. 00
03

N
W E
S
1 TÚNEL
9 SENDERO
10 CLARO
5 SENDERO
2 PANTANO
8 SENDERO
4 TÚNEL
6 SENDERO
7 SENDERO
3 PUEBLO

David Higham

en una descripción textual. No obstante, este aumento de popularidad de los juegos de aventuras se puede atribuir casi con total seguridad al mayor atractivo visual que otorgan los gráficos y, si bien algunos recientes juegos para micros sólo utilizan imágenes sencillas para mejorar el texto, otros intentan dar mayor relevancia a la vertiente visual.

En nuestro proyecto de programación analizaremos las técnicas que se emplean en la programación de un juego de aventuras. A lo largo del proyecto se le irán proporcionando secciones de un listado para un juego de aventuras llamado *Digitaya*, que irán construyendo un programa completo. En este juego, el participante asume el papel de un agente "electrónico" al que se le encarga la tarea de descender al interior de un microordenador para localizar y rescatar al misterioso Digitaya de las garras de la máquina. A lo largo del camino se presentan muchas dificultades y acechan muchos peligros, y el jugador habrá de valerse de todos sus conocimientos sobre los ordenadores para escapar ileso. El programa, en la medida de lo posible, está escrito en BASIC "estándar", ofreciéndose los *complementos* en los casos necesarios. Por lo tanto, siempre y cuando se disponga de suficiente capacidad de memoria, el programa funcionará en su ordenador. Puesto que vamos a estudiar con todo detalle las diversas técnicas de programación, sería difícil no revelar muchos de los secretos del juego, y ello iría en detrimento, en cierta medida, del placer de jugar con él cuando estuviera acabado. Por consiguiente, construiremos, juntamente con *Digitaya*, un juego paralelo, más breve, llamado *El bosque encantado*, con el que demostraremos las técnicas y los algoritmos utilizados para construir el juego más extenso.

El punto de partida para el diseño de nuestro juego de aventuras es el trazado de un mapa del mundo fantástico que estamos imaginando. En este mapa hemos de marcar los diversos escenarios del mundo, la posición de todos los objetos que se puedan hallar en los mismos, y dar importancia a aquellos escenarios que se consideren "especiales". La mayoría de los escenarios del mapa simplemente permitirán que el jugador entre y salga de ellos y recoja o abandone cualquiera de los objetos que haya en ellos. Los escenarios especiales pueden ser peligrosos (un pantano o un lugar donde habita un dragón) o pueden exigir la realización de una serie de acciones especiales antes de que uno pueda penetrar en ellos o salir de los mismos.

La mejor forma de empezar a hacer un mapa consiste en considerar en líneas generales cuántos escenarios se necesitan para el juego. *El bosque encantado* tiene 10 escenarios y se diseñó en una cuadrícula de 5 × 5 (como refleja la ilustración), mientras que *Digitaya* posee alrededor de 60 escenarios y se diseñó en una cuadrícula de 10 × 10.

Los cuadrados de la cuadrícula inicialmente no tienen números y el diseñador comienza por rellenar el mapa con escenarios. En el mapa de *El bosque encantado* hay un sendero, dos túneles, un pantano, un claro y un pueblo. También están marcadas, en la parte inferior de los cuadrados en los que están ubicados, las posiciones de varios objetos. Aquellos escenarios señalados con un asterisco (*) son "especiales" y se los tratará de forma distinta que al resto de los mismos.

Una vez terminado el trazado podemos numerar cada uno de los escenarios. La única consideración especial que hemos tenido en cuenta al elegir el número de escenarios es que todos los especiales se han numerado primero. El orden por el cual se numeren los otros carece de importancia, pero una vez que se han seleccionado los números es importante no modificarlos con posterioridad.

Programación de los datos

La primera tarea de programación es convertir la información del mapa en datos para el programa. Existen muchas formas de hacer esto, pero lo que haremos aquí es utilizar dos matrices unidimensionales para retener los datos. La primera matriz, ES$(), retiene descripciones de cada *es*cenario. Por ejemplo, para el escenario 7, ES$(7) contendrá "en un sendero". Cuando posteriormente los datos se utilicen en el programa para describir un escenario irán precedidos por las palabras "Ud. se halla".

La segunda matriz, SL$(), retiene datos relativos a los posibles movimientos que se puedan efectuar desde un escenario (*sal*idas). Nuestros dos juegos se limitan a cuatro direcciones: Norte, Este, Sur y Oeste. SL$() proporciona información acerca del número de escenario al cual uno se traslada para cada una de las cuatro direcciones. Los datos están almacenados como una serie compuesta por ocho dígitos. El número de escenario para cada dirección se entra por el orden NESO, utilizando un número de dos dígitos para cada dirección.

Por ejemplo, el escenario 7 tiene salidas al Norte, al Sur y al Oeste, pero ninguna hacia el Este. Los dos primeros dígitos de SL$(7) son 08 (no 8), que indica que hacia el Norte está el escenario 8. El segundo par de dígitos, 00, indica que en esta dirección no hay ninguna salida (Este). Los pares de dígitos 03 y 06 representan los escenarios que se encuentran hacia el Sur y hacia el Oeste del emplazamiento 7. Utilizando este sistema se podrían especificar hasta 39 escenarios; si se necesitaran más de 39, entonces los datos para SL$() se habrían de entrar como grupos de tres dígitos.

Los tres objetos de *El bosque encantado* se leen en otra matriz, IV$(,). Esta matriz bidimensional lleva el registro de la posición de cada objeto mientras el mismo se va trasladando a través del bosque. Cada objeto posee una descripción y su emplazamiento inicial en el mapa. Por ejemplo, IV$(C,1) es ESCOPETA, y su posición al comienzo del juego viene dada por IV$(C,2). A medida que, en el transcurso del juego, los objetos se vayan trasladando de un lugar a otro, los elementos de posición de la matriz se irán actualizando.

Al final de los datos del mapa, en nuestros dos listados hay otros datos. Se trata de una "suma de control" y se proporciona para asegurar que los datos de dirección se hayan digitado correctamente. Ello se realiza calculando un total actualizado de los valores de los datos, que se cotejan con la suma de control. Si éstos no coinciden, entonces se ha producido un error y la ejecución del programa se interrumpirá. Usted observará que en *Digitaya* se utilizan dos sumas de control. Ello se debe a que la suma total de los datos de dirección es demasiado grande como para retenerla fácilmente en una sola suma de control, de modo que se calcula por separado un total para los cuatro dígitos izquierdos y los cuatro derechos. En el próximo capítulo del proyecto diseñaremos rutinas para manipular y visualizar los datos de este mapa.

Digitaya

```
6090 REM **** S/R LEER DATOS MATRIZ ****
6100 REM ** LEER INVENTARIO **
6110 DIM IV$(8,2)IC$(4)
6120 FOR C=1T08
6130 READ IV$(C,1),IV$(C,2)
6140 NEXT C
6150 :
6160 REM ** LEER DATOS ESCENARIOS & SALIDAS **
6170 DIM ES$(55),SL$(55)
6180 C1=0:C2=0: REM INICIALIZAR SUMAS DE CONTROL
6190 FOR C=1T054
6200 READ ES$(C),SL$(C)
6210 C1=C1+VAL(LEFT$(C),4))
6220 C2=C2+VAL(RIGHT$(SL$(C),4))
6230 NEXT C
6240 READ CA:IFCA<>C1 THEN PRINT"ERROR SUMA DE CONTROL":
     STOP
6250 READ CA:IFCB<>C2 THEN PRINT"ERROR SUMA DE CONTROL":
     STOP
6260 RETURN
6270 REM **** DATOS DE INVENTARIO ****
6280 DATA NUMERO DE DIRECCION,45,LLAVE,34,ESCUDO
     LASER,25
6290 DATA BILLETE AL TRIESTADO,26,TARJETA DE CREDITO DE
     DATOS,28
6300 DATA DIGITAYA,30,LIBRO DE CODIGOS,19,DISPOSITIVO
     ACTIVADOR DEL BUFFER,13
6310 :
6320 REM **** DATOS ESCENARIOS & SALIDAS ****
6330 DATA EN LA TOMA DEL TELEVISOR,00000000
6340 DATA EN LA PUERTA PARA EL USUARIO,00090100
6350 DATA EN LA PUERTA PARA CASSETTE,00110000
6360 DATA EN LA PUERTA PARA PALANCA DE MANDOS,
     00130000
6370 DATA EN UN DISPOSITIVO TRIESTADO,00170000
6380 DATA EN LA UNIDAD ARITMETICA Y LOGICA,00310016
6390 DATA EN EL ACCESO A LA MEMORIA,00490000
6400 DATA EN LA VIA DE E/S,09000001
6410 DATA EN LA VIA DE E/S,10000802
6420 DATA EN LA VIA DE E/S,11000900
6430 DATA EN LA VIA DE E/S,12001003
6440 DATA EN LA VIA DE E/S,13531100
6450 DATA EN LA VIA DE E/S,14001204
6460 DATA EN LA VIA DE E/S,15001300
6470 DATA EN LA VIA DE E/S UNA SEÑAL DICE "S OUT H",
     00001400
6480 DATA EN EL REGISTRO DE DATOS,00061700
6490 DATA EN UNA VIA DE 8 PISTAS,16001805
6500 DATA EN UNA VIA DE 8 PISTAS,17001900
6510 DATA EN UNA VIA DE 8 PISTAS,18002000
6520 DATA EN UNA VIA DE 8 PISTAS,19292100
6530 DATA EN UNA VIA DE 8 PISTAS,20282200
6540 DATA EN UNA VIA DE 8 PISTAS,21272300
6550 DATA EN UNA VIA DE 8 PISTAS,22262400
6560 DATA EN UNA VIA DE 8 PISTAS,23250000
6570 DATA EN LA MATRIZ DE CARACTERES,26360024
6580 DATA EN LO ALTO DE LA MEMORIA,27352523
6590 DATA EN LA MITAD DE LA MEMORIA,28342622
6600 DATA EN LA MITAD DE LA MEMORIA,29332721
6610 DATA ABAJO EN LA MEMORIA,00542820
6620 DATA EN LA GUARIDA DEL ACUMULADOR,00000600
6630 DATA EN UN LARGO CORREDOR,00420006
6640 DATA EN UN REGISTRO INDICE,31000000
6650 DATA ABAJO EN LA MEMORIA,54403428
6660 DATA EN LA MITAD DE LA MEMORIA,33393527
6670 DATA MUY EN LO ALTO DE LA MEMORIA,34383626
6680 DATA EN LA MATRIZ DE CARACTERES,35370025
6690 DATA EN UNA TABLA VECTORIAL ALEATORIA,00000000
6700 DATA EN LO ALTO DE LA MEMORIA CON VISTA A UNA VIA,39003735
6710 DATA EN LA MITAD DE LA MEMORIA,40003834
6720 DATA EN LA MEMORIA — HACIA EL ESTE HAY UN ACCESO,
     41003933
6730 DATA ABAJO EN LA MEMORIA,00004054
6740 DATA EN UN CORREDOR,00430031
6750 DATA EN UN CORREDOR,00440042
6760 DATA EN UN CORREDOR,00004543
6770 DATA EN EL REGISTRO DE DIRECCIONES,00004600
6780 DATA EN UNA VIA DE 16 PISTAS,45004700
6790 DATA EN UNA VIA DE 16 PISTAS,46004800
6800 DATA EN UNA VIA DE 16 PISTAS,47004900
6810 DATA EN UNA VIA DE 16 PISTAS SE VISLUMBRA HACIA EL OESTE UN ENORME
     PORTALON,48005007
6820 DATA EN UNA VIA DE 16 PISTAS,49005100
6830 DATA EN UNA VIA DE 16 PISTAS,50005200
6840 DATA EN UNA VIA DE 16 PISTAS,51000000
6850 DATA EN UN VECTOR A LA MEMORIA,00290012
6860 DATA ABAJO EN LA MEMORIA,00413329
6870 REM ** DATOS SUMAS DE CONTROL **
6880 DATA 100169,103973
```

El bosque encantado

```
6000 REM **** LEER DATOS OBJETOS Y MAPA ****
6010 DIM IV$(3,2),ES$(10),SL$(10),IC$(2)
6020 FOR C=1 TO 3
6030 READ IV$(C,1),IV$(C,2)
6040 NEXT C
6050 :
6060 FOR C=1 TO 10
6065 READ ES$(C),SL$(C)
6070 CC=CC+VAL(SL$(C)):REM TOTAL SUMA CONTROL
6080 NEXT C
6090 :
6100 READ CD:IFCD<>CC THENPRINT"ERROR EN SUMA DE CONTROL":STOP
6110 :
6120 REM ** DATOS OBJETOS **
6130 DATA ESCOPETA,10,FAROL,9,LLAVE,5
6140 :
6150 REM ** DATOS MAPA **
6160 DATA JUNTO A LA ENTRADA DE UN TUNEL,00000900
6170 DATA EN UN PANTANO,00000000
6180 DATA EN UN PUEBLO,07000000
6190 DATA JUNTO A LA ENTRADA DE UN TUNEL,05060000
6200 DATA EN UN SENDERO,00020400
6210 DATA EN UN SENDERO,02070004
6220 DATA EN UN SENDERO,08000306
6230 DATA EN UN SENDERO,09000702
6240 DATA EN UN SENDERO,01100800
6250 DATA EN UN CLARO,00000009
6260 REM ** DATOS SUMA DE CONTROL **
6270 DATA 32253121
6280 RETURN
```

Complementos al BASIC

Nuestros listados se escribieron para el Commodore 64. Para el Spectrum y el BBC Micro es necesario introducir estas modificaciones:

Spectrum:

Para el listado de *Digitaya*, realice estos cambios:

```
6110 DIM V$(8,2,24):DIM I$(4,24)
6170 DIM L$(55,52):DIM E$(55,8)
6200 READ L$(C),E$(C)
6210 LET C1=C1+VAL(E$(C)(TO 4))
6220 LET C2=C2+VAL(E$(C)(LEN(E$(C))-3 TO))
```

Además, todos los valores DATA se deben encerrar entre comillas, con la excepción de la suma de control de la línea 6880. En el listado de *El bosque encantado* introduzca estas modificaciones:

```
6010 DIM V$(3,2,5):DIM L$(10,22)
6015 DIM E$(10,8):DIM I$(2,5)
6030 READ V$(C,1)V$(C,2)
6065 READ L$(C),E$(C)
6067 LET CC=0
6070 LET CC=CC+VAL(E$(C))
```

Encierre entre comillas los valores DATA, a excepción de la suma de control de la línea 6270.

BBC Micro:

En el listado de *El bosque...* agregue esta línea:

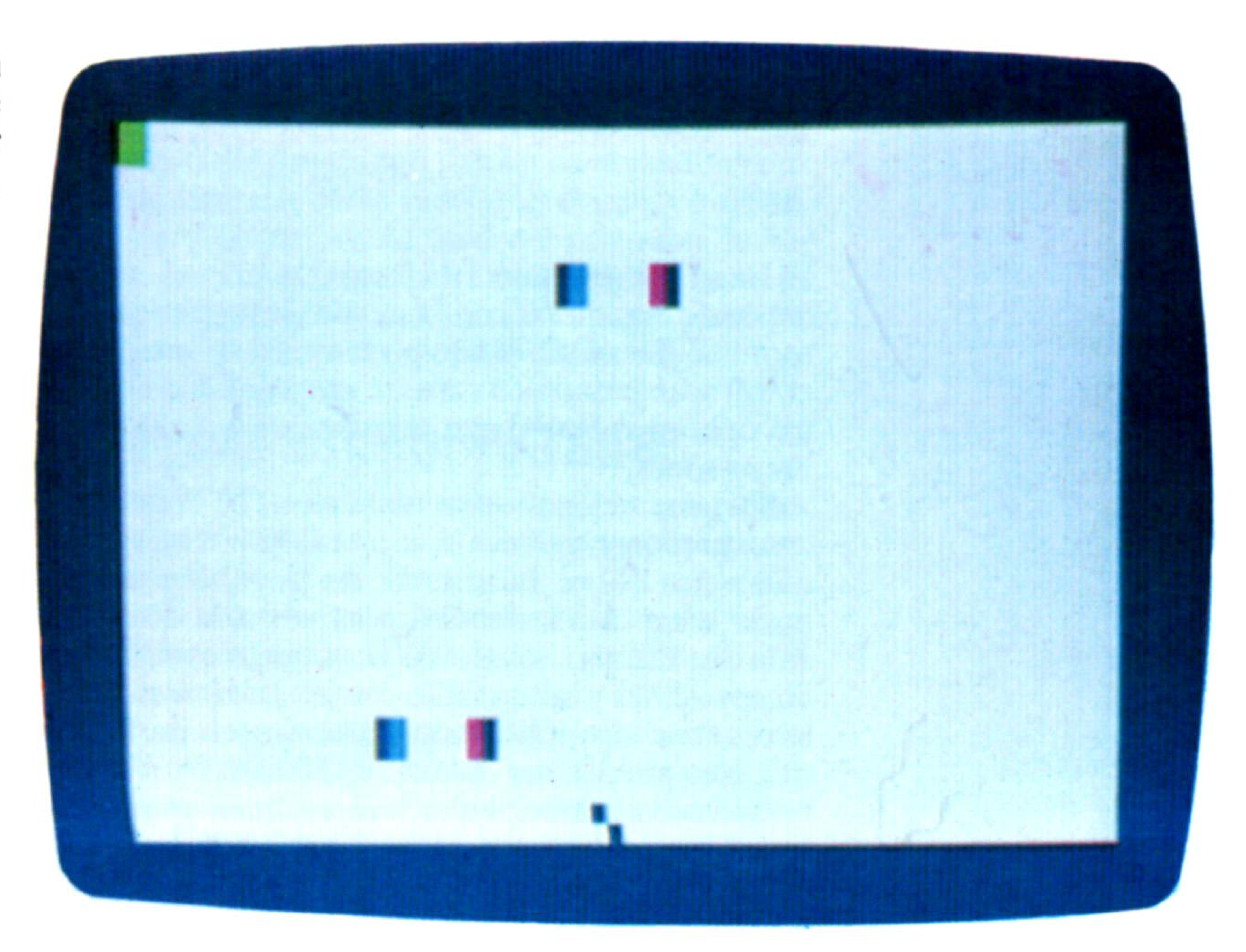
```
6067 CC=0
```

En el de *Digitaya* no se precisan modificaciones.

Slalom

Los juegos deportivos ejercen gran atracción en la mayoría de los usuarios. Presentamos un «juego de invierno» en una versión para el microordenador Dragon

¡Dispóngase a practicar los deportes de invierno sin correr el riesgo de romperse una pierna! Láncese desde lo alto de la pista e intente pasar por el mayor número posible de puertas sin chocar con los palos. Para cambiar de dirección pulse cualquier tecla.

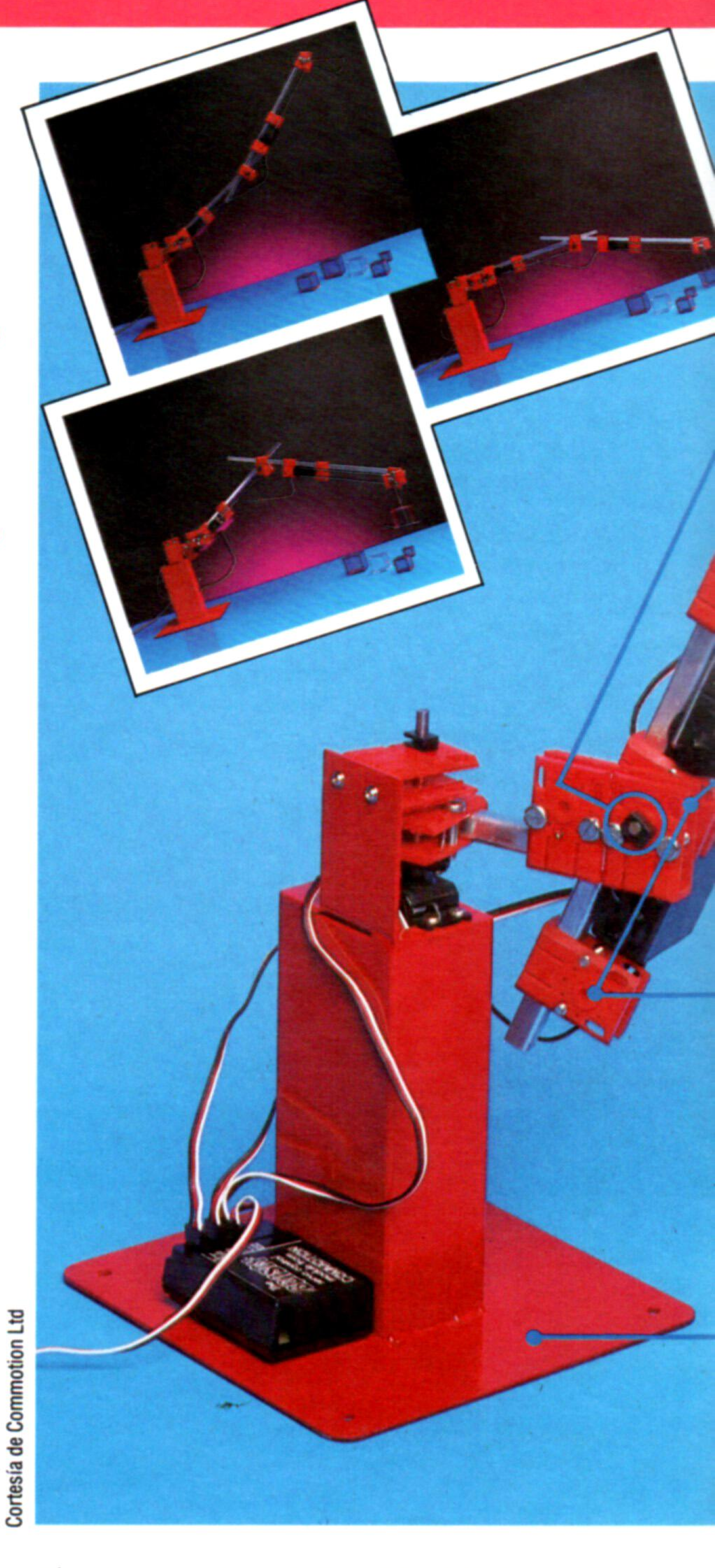


```
10 REM **********
20 REM * SLALOM *
30 REM **********
34 REM
35 REM INICIALIZACION
36 REM
38 REM TABLA DE POSICIONES
39 REM DEL ESQUIADOR
40 DIM S$(1)
50 FOR I=1 TO 32
60 E$=E$+CHR$(207)
70 NEXT I
79 REM ESQUIADOR HACIA LA IZQUIERDA
80 S$(0)=CHR$(201)
89 REM ESQUIADOR HACIA LA DERECHA
90 S$(1)=CHR$(198)
99 REM FONDO BLANCO
100 CLS 5
108 REM DIRECCION INICIAL:
109 REM                  IZQUIERDA
110 D=-1
118 REM POSICION INICIAL
119 REM DEL ESQUIADOR
120 J=16
129 REM DISEÑO DE LAS PUERTAS
130 P$=CHR$(181)+CHR$(207)+CHR$(207)+CHR$(170)
134 REM
135 REM BUCLE PRINCIPAL
136 REM
140 FOR K=1 TO 300
145 REM CALCULO DE COORDENADAS
146 REM DEL ESQUIADOR
150 Y=INT (J/32)*2
160 X=(J-16*Y)*2
165 REM ESQUIADOR A LA ALTURA DE
166 REM UNA PUERTA?
170 IF K>=16 AND (K-5)/10=INT((K-5)/10) THEN
    GOSUB 330
179 REM FIJACION DE UNA PUERTA?
180 IF K<284 AND K/10=INT(K/10) THEN GOSUB
    350
189 REM MOVIMIENTO DEL ESQUIADOR
190 IF INKEY$<>"" THEN D=-D
200 J=J+D
210 IF J<2 THEN J=2
220 IF J>29 THEN J=29
230 PRINT @ 511,E$;
240 PRINT @ J,S$(D/2+0.5);
250 NEXT K
254 REM
255 REM FIN DEL RECORRIDO
256 REM
260 PRINT @ 164,"PUERTA(S) SALTADA(S) :";T;
270 PRINT @ 229,"OTRA BAJADA ?";
280 D$=INKEY$
290 IF D$="" THEN 280
300 IF D$<>"N" THEN RUN
310 CLS
320 END
324 REM
325 REM PUERTA SALTADA?
326 REM
330 IF POINT (X-2,Y)<>0 OR POINT (X+4,Y)<>3
    THEN IF POINT (X-4,Y)<>0 OR POINT
    (X+2,Y)<>3 THEN T=T+1:SOUND 1,1
340 RETURN
344 REM
345 REM FIJACION DE UNA PUERTA
346 REM
350 P1=RND(3)-2
360 P=P-6*P1
370 IF P<482 THEN P=488
380 IF P>506 THEN P=500
390 PRINT @ P,P$;
400 RETURN
```

© Sybex

Un robot en casa

Poco a poco están empezando a aparecer en el mercado robots de precio económico: el Beasty ha sido uno de los primeros



Cortesía de Commotion Ltd

El Beasty se suministra en forma de *kit*, y el usuario lo puede ensamblar con la ayuda de un par de destornilladores. Con el equipo también se suministra el software basado en cassette que contiene el sistema operativo Robol que se utiliza para controlar el brazo-robot.

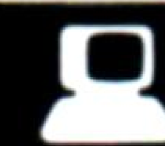
Se proporcionan dos manuales. El folleto de construcción comienza con una larga introducción que versa sobre la historia de la robótica, para pasar luego a los detalles relativos a la construcción. Al usuario novato tal vez la gran cantidad de componentes y las un tanto complejas instrucciones le resulten algo intimidantes; se ofrecen ilustraciones, que pueden ser de mayor utilidad. No obstante, incluso el principiante será capaz de montar el brazo correctamente, si bien conseguirlo le llevará un poco de tiempo. De momento, el Beasty no se puede adquirir ya montado, si bien el fabricante, Commotion, afirma que lo puede ofrecer armado si así se le solicita.

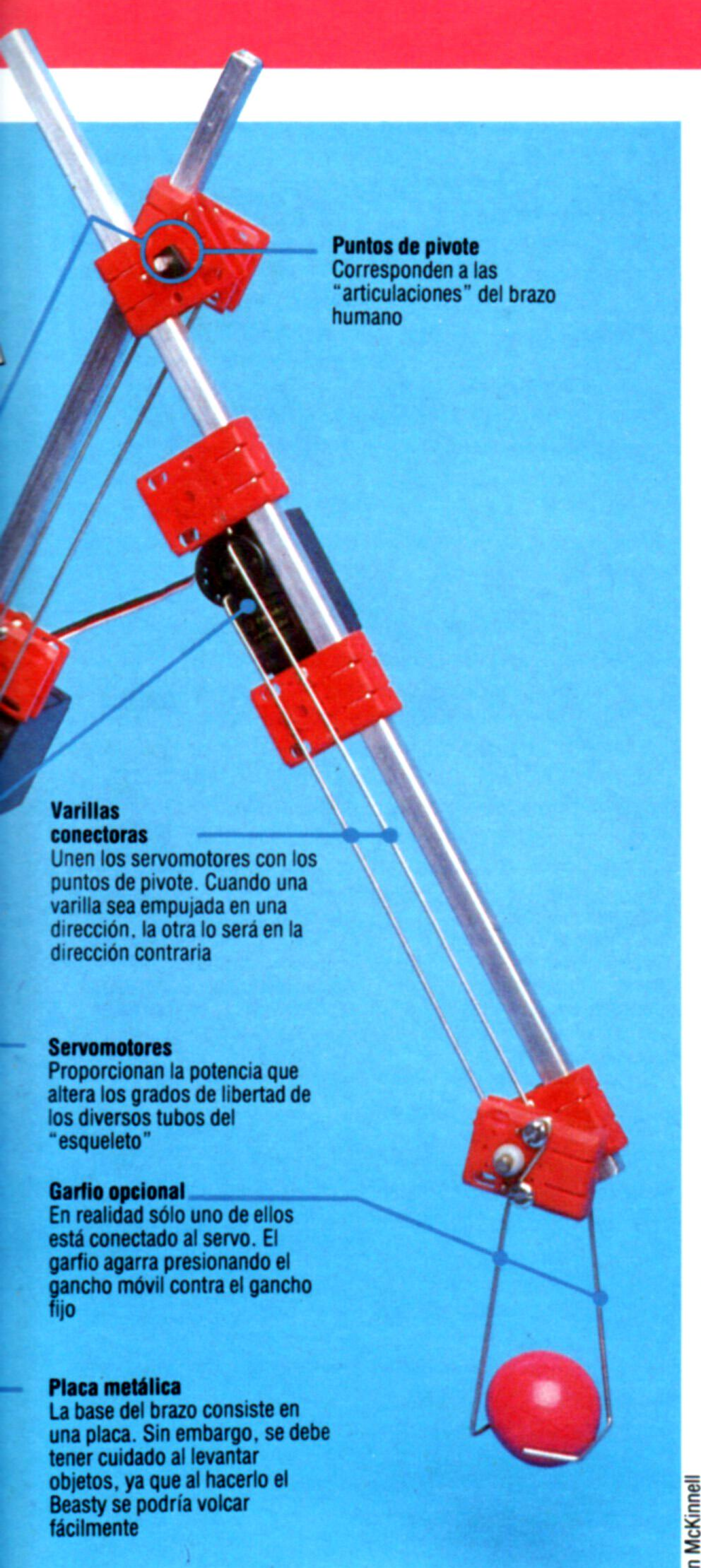
Una vez montado, el Beasty se compone de una base que soporta una juntura que posibilita el movimiento lateral. Esta juntura lleva conectada una corta varilla de aluminio, que está unida a la parte superior del brazo mediante una segunda juntura. Una tercera juntura conecta el antebrazo. Estas junturas están alimentadas por servomotores, cada uno de los cuales controla dos cortas varillas rígidas que están conectadas al "esqueleto" del brazo. Cuando un servomotor gira, tira hacia él de una de las varillas y empuja a la otra en la dirección contraria, haciendo girar, por consiguiente, la juntura y moviendo el brazo. Un servomotor opera traduciendo impulsos digitales en movimiento. El motor recibe una serie de impulsos a una determinada frecuencia y el procesador interpreta estos impulsos como un ángulo de movimiento. Mientras la frecuencia permanezca constante, el motor mantendrá el brazo en su posición actual; un cambio en la frecuencia de los impulsos le indica al procesador que se requiere un nuevo ángulo y entonces el brazo se moverá.

Los servos FP128 utilizados en el Beasty pueden generar 3,5 kg/cm de empuje. Esto significa que, en 1 cm a lo largo de un eje, el servo puede levantar 3,5 k, mientras que lo largo de 10 cm puede levantar 350 g. Éste es un punto importante a tener en cuenta cuando se levantan pesos; obviamente, el servo del "hombro", al estar más alejado del peso a levantar, será el que soportará la mayor tensión.

El servoprocesador está alojado en una pequeña caja negra que no se halla unida al brazo propiamente dicho. Esta caja posee conectores para conectar hasta cuatro servomotores (la cuarta conexión es para un motor opcional que se puede utilizar para operar un "garfio" o cualquier dispositivo de agarre similar en el extremo del antebrazo). También hay un conector de entrada que se conecta en interface con la puerta para el usuario del BBC, un cable de potencia que se enchufa en el conector de potencia auxiliar del ordenador.

Una vez cargado el software desde cassette, la pantalla visualiza una indicación que le recuerda al usuario que el sistema está en modalidad Edit. En Robol, una línea de programa se compone de un número de línea, una construcción y una serie de números, cada uno de los cuales corresponde a una de las cuatro opciones de servomotores. Si la línea contiene la instrucción MOVE (mover), los números corresponden a la frecuencia de impulsos que man-





Ian McKinnell

Servomotores
El Beasty estándar viene equipado con tres servomotores conectados a una caja interface. Los servomotores son motores muy utilizados en robótica debido a la forma en que proporcionan una fuerza constante o "momento de torsión" correspondiente a la frecuencia de la señal digital suministrada

tienen a los servos en sus posiciones en curso. El usuario puede alterar estos números y mientras el sistema esté en modalidad Edit el servomotor que se esté regulando se moverá a medida que se efectúen las modificaciones, lo que permite posicionar el brazo tal como lo desee el usuario. Cuando el operador se sienta satisfecho respecto a las posiciones de cada uno de los motores, se debe pulsar Return, después de lo cual se visualiza una nueva línea en Robol y se puede programar la siguiente serie de movimientos.

Si se pulsa la tecla de función F0 se puede hacer que el brazo lleve a cabo una secuencia completa de movimientos. El programa se puede comenzar a ejecutar a partir de cualquier línea pulsando primero F1 y después F0 (ésta produce la ejecución desde el principio del programa), o bien cambiando el número de línea en curso mediante el empleo de las teclas para el cursor. Al final del programa la secuencia de acciones se repetirá de forma automática. Si el usuario deseara detener el programa, deberá cambiar una instrucción MOVE por STOP.

Demoras cronometradas

Durante la ejecución de una serie de instrucciones MOVE se puede hacer que el brazo haga una pausa mediante la incorporación de una instrucción WAIT (esperar), seguida de un número. Ésta trabaja accediendo a la patilla TIMER 1 de la puerta para el usuario, generando una interrupción. Dado que el reloj trabaja en unidades de 1/100 (centésimas) de segundo, WAIT 100 producirá una demora de 1 s antes de que se lleve a cabo la siguiente instrucción.

La acción del brazo se puede acelerar enormemente cambiando la instrucción MOVE por JUMP (saltar). También se incluyen dos sentencias de cronometraje: JDELAY y MDELAY. El Beasty posee una demora incorporada que se produce antes de la ejecución de cada línea. Ésta posee un valor por defecto de 20 (es decir, 1/5 s), pero este valor se puede alterar utilizando JDELAY para sentencias JUMP y MDELAY para instrucciones MOVE.

El sistema operativo Robol es fácil de utilizar, y programar el brazo para que realice movimientos complejos es una tarea sencilla que ocupa sólo minutos. El manual de operaciones es breve pero perfectamente adecuado, aunque los programadores avanzados quizá encuentren que la información que proporciona para una programación más compleja es insuficiente. El Beasty se puede controlar desde BASIC utilizando el programa *Driver*; éste accede a la puerta para el usuario del BBC Micro de forma muy similar a los ejemplos que hemos ofrecido en nuestro apartado de *Bricolaje*.

Commotion ha incluido, asimismo, un corto programa que permite efectuar copias del software Robol. Lamentablemente, la mayoría de las unidades de disco BBC emplean el conector de potencia auxiliar del BBC Micro, por lo cual no se pueden conectar al mismo tiempo el Beasty y una unidad de disco.

No obstante, a pesar de estos nimios detalles el Beasty es, ciertamente, una valiosa introducción al campo de la robótica. Se podría decir que éste es un dispositivo en espera de una aplicación, dado que en realidad no se puede afirmar que el brazo-robot sea auténticamente útil y es bastante probable que sólo lo adquieran los aficionados más entusiastas.

EL BEASTY

SOFTWARE
Robol y rutinas activadoras en cassette.

DOCUMENTACION
El Beasty viene con un manual de montaje y una guía de programación.

VENTAJAS
Constituye una introducción a la robótica fácil de utilizar y de precio económico.

DESVENTAJAS
La guía de montaje adolece de falta de claridad y existe una carencia de aplicaciones obvias para el brazo, aunque Commotion ha prometido algunas mejoras que ampliarán sus usos.

Por el laberinto

Ahora desarrollaremos un programa «inteligente» que conducirá a través de un laberinto el vehículo que hemos creado

La primera etapa en la construcción de un laberinto es decidir dónde lo realizaremos. Podría ser en la superficie de una mesa o sobre el suelo. El área designada se dividirá entonces en cierto número de cuadrados, dependiendo el tamaño de cada uno de éstos de las dimensiones del vehículo que se utilizará para cruzar el laberinto. Cada cuadrado debe ser lo suficientemente grande como para permitir que el vehículo pivote 360° en el interior de un único cuadrado. Entonces se puede marcar la superficie a modo de cuadrícula. Para conformar el laberinto se pueden colocar sobre la superficie objetos tales como libros, tazas o pequeños trozos de madera.

El programa exige que se especifiquen las dimensiones del laberinto y las posiciones de los cuadrados que estén ocupadas y aquellas que estén libres. El método más sencillo de hacer esto consiste en utilizar un código binario: 1 para indicar que un cuadrado está parcial o totalmente ocupado por un objeto, 0 para señalar que está vacío. Para que no sea necesario entrar los datos relativos al laberinto cada vez que se ejecute el programa, esta información se debe escribir como una serie de sentencias DATA. Los cuatro datos finales son las coordenadas del punto de partida y del punto de llegada. Podemos imaginar que el origen de coordenadas está en la esquina superior izquierda, siendo la fila superior la fila 0, y siendo la columna situada más a la izquierda la columna 0. Este laberinto es el correspondiente a las siguientes sentencias DATA:

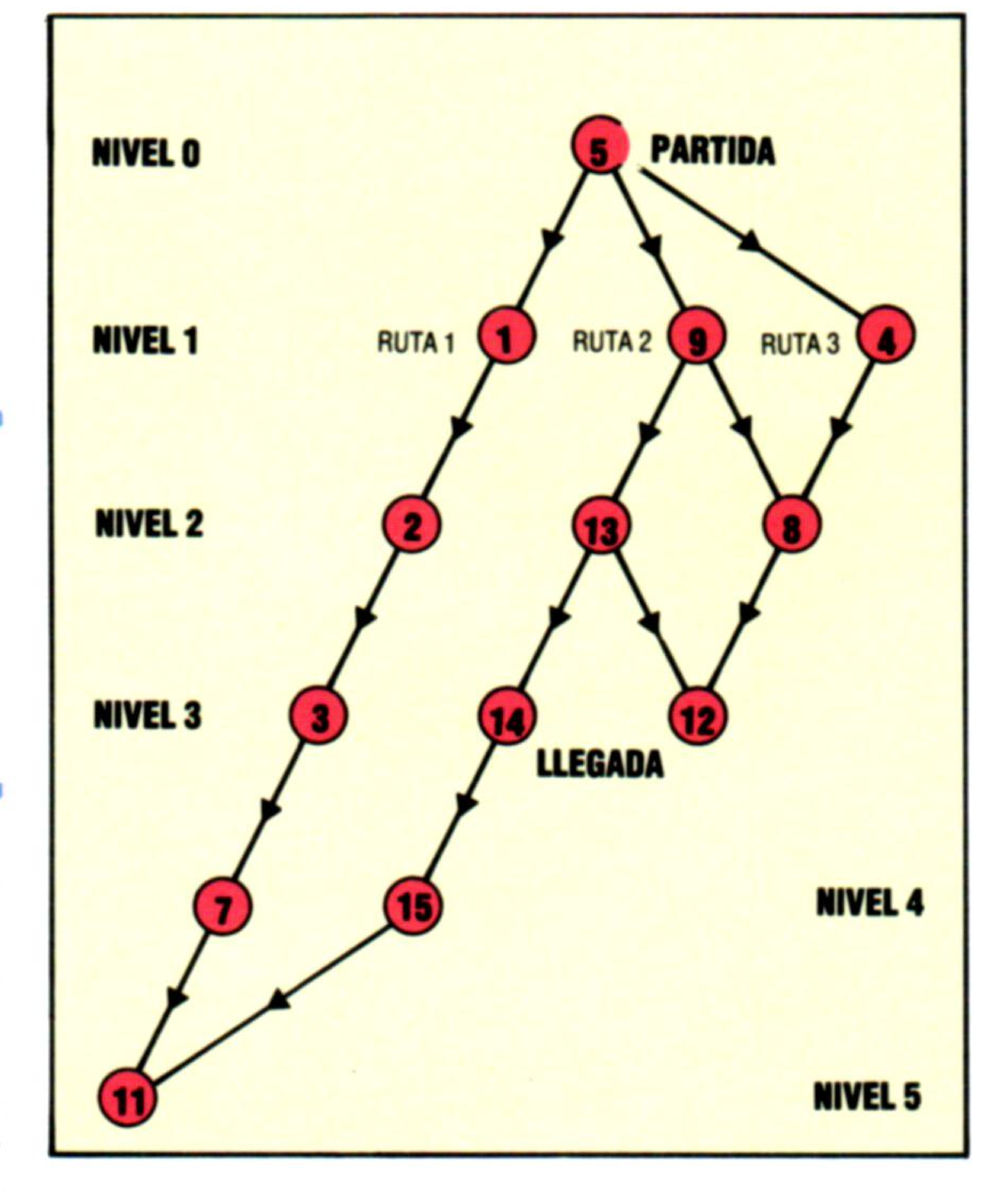
```
DATA 4,4:REM DIMENSIONES DEL LABERINTO
DATA 1,0,0,0,0,0,1,0
DATA 0,0,1,0,0,0,0,0
DATA 1,1:REM COORDS DE PARTIDA
DATA 2,3:REM COORDS DE LLEGADA
```

Encontrar una ruta a través del laberinto no presenta excesivas dificultades. Podemos diseñar un programa que trace un camino desde el punto de partida, retrocediendo en los callejones sin salida y volviendo hacia atrás los pasos necesarios hasta llegar finalmente al punto de llegada. La ruta hallada (sin los rodeos de los callejones sin salida) puede o no ser la ruta más corta posible. Si deseáramos hallar la *mejor* ruta entre los puntos de partida y de llegada, entonces deberíamos adoptar un procedimiento que pruebe todos los posibles caminos entre los dos puntos. Vale la pena señalar que nuestro programa interpreta que la "mejor ruta" es la que utiliza la menor cantidad de cuadrados.

Podemos simplificar la tarea de probar cada una de las rutas creando una estructura en los datos del laberinto que represente las relaciones existentes entre los cuadrados. La estructura de datos que mejor se presta para esta aplicación es un árbol jerárquico. Empezando con el punto de partida como la "raíz" del árbol, podemos construir una segunda generación de cuadrados (o "nudos") que están a una distancia de un cuadrado de la raíz. A partir de esta segunda generación de nudos se puede construir una tercera generación, y así sucesivamente. Podemos dibujar un árbol para cualquier laberinto numerando cada cuadrado y siguiendo la regla de que los descendientes de cualquier nudo se dibujan de izquierda a derecha por el orden Norte, Este, Sur y Oeste del nudo principal del laberinto.

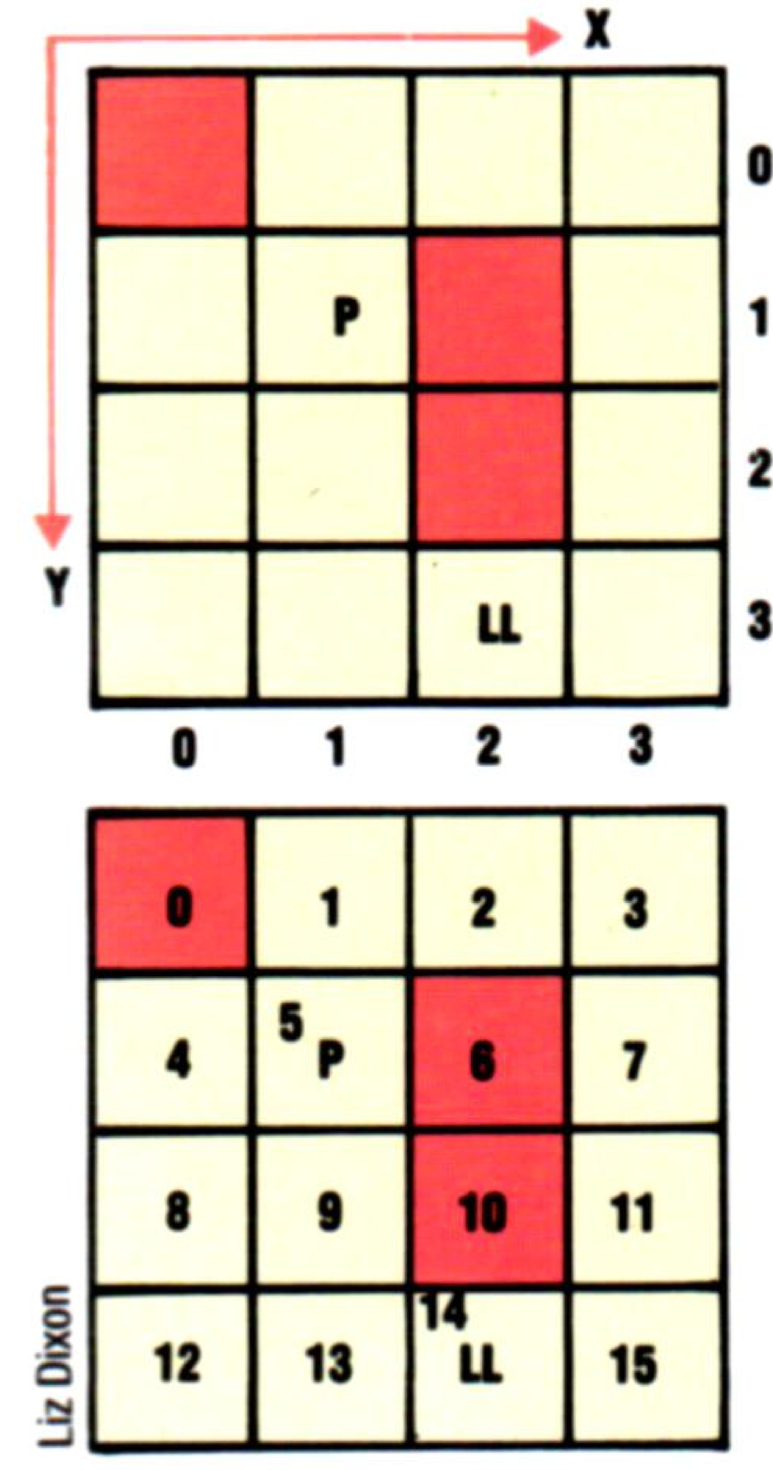
Este sencillo laberinto se puede resolver de cinco formas, sin desandar los pasos dados. En la ilustración superior se muestran tres posibles soluciones, como rutas a través del árbol y como rutas verdaderas por el laberinto. Para nosotros es evidente que la ruta 2 es la más corta, pero ello se debe a que podemos evaluar el árbol lateralmente, es decir, podemos considerarlo como un todo. El ordenador debe resolver el árbol de forma lineal, tomando sistemáticamente cada uno de los caminos posibles hasta hallar el nudo final o llegar a un callejón sin salida. En el primer caso se debe llevar un registro de la ruta de éxito; en el segundo, antes de volver a empezar desde el nudo raíz se debe señalar el camino tomado como un callejón sin salida. El programa continuará recorriendo el árbol hasta haber probado todas las ramas que parten del nudo raíz.

El BASIC no se presta fácilmente para tratar algoritmos de este tipo y con frecuencia la programación puede resultar torpe y difícil de manejar. Len-



Estructura de árbol
Para encontrar la mejor ruta a través del laberinto se debe construir un "árbol" que represente las relaciones entre los cuadrados. Cada nudo se considera de uno en uno, creando niveles de nudos inferiores. Los nudos del nivel 1 están a un cuadrado de distancia de la partida; los nudos del nivel 2 están a dos cuadrados de distancia, etc. Es bastante directo dibujar el árbol, pero implementar esta estructura en BASIC es más difícil

Conducción en doble sentido
Este programa para resolver un laberinto interpreta a éste de dos maneras. Dado que el laberinto se lee a partir de sentencias de datos, está almacenado en una matriz bidimensional, reteniéndose asimismo inicialmente los puntos de partida y de llegada como coordenadas. Con el fin de resolver el laberinto, el programa debe tratar cada uno de los cuadrados del laberinto como el "nudo" de un árbol. En vez de utilizar el sistema de coordenadas inicial, cada cuadrado se numera, por consiguiente, por orden, empezando en el rincón superior izquierdo del laberinto



guajes como el LOGO y el ALGOL están mucho mejor dotados para cumplir esta misión. En BASIC tenemos que llevar a cabo dos tareas fundamentales: en primer lugar, debemos obtener nuestro árbol a partir de los datos del laberinto, tal como se le presentan al programa. Y por cada cuadrado del laberinto debemos tener cuatro apuntadores que indiquen qué cuadrado hay en cada una de las cuatro direcciones. Para almacenar este sistema de señaladores, lo más indicado es una matriz bidimensional, TR(N,D), donde N es el número del cuadrado y D es la dirección de 1 a 4. Por consiguiente, en nuestro laberinto simple, TR(9,1) sería 5: el cuadrado que se halla al norte del cuadrado 9. Cuando el cuadrado de una dirección determinada no está libre, o si hay una frontera del laberinto, ello se puede marcar mediante un valor especial, por ejemplo -1.

A medida que se va recorriendo el árbol, la ruta tomada se almacena en una seudopila, implementada utilizando una matriz unidimensional y una variable, D, para indicar el siguiente espacio disponible en la pila. El camino más corto que se encuentre también se almacena en una matriz unidimensional, con el número de pasos para la ruta almacenado en el primer elemento de la matriz.

Cuando el programa ha recorrido su camino a través del árbol, se retendrá un registro de la mejor ruta en forma de una serie de números de cuadrados. Suponiendo que originalmente el vehículo estuviera orientado hacia el Norte en el cuadrado de partida, se lo puede dirigir utilizando las relaciones matemáticas simples entre la dirección a recorrer y la diferencia entre dos números de cuadrados consecutivos en la matriz de ruta. Por ejemplo, en nuestro sencillo laberinto, una diferencia de +4 indicaría norte, −4 indicaría sur, y así sucesivamente. Debemos entonces calcular el ángulo a describir para cambiar la dirección, antes de seguir adelante hacia otro cuadrado. Dado que el vehículo emplea motores eléctricos CD simples, los ángulos de giro y las distancias recorridas se gobiernan mediante el período de tiempo durante el cual está encendida una determinada combinación de motores. Para hacer un uso práctico del programa es necesario realizar algunos experimentos iniciales para determinar los intervalos de tiempo requeridos para describir un ángulo de 90° y avanzar un cuadrado. Esta información se debe entrar en las variables AF y FF, respectivamente. La versión para el BBC exige unidades de una centésima de segundo, y la versión para el Commodore 64 requiere unidades de un sesentavo de segundo.


Ian McKinnell

Solución al laberinto

```
900 REM ************************************
910 REM ************************************
920 REM **                                **
930 REM **        LABERINTO CBM 64        **
950 REM **                                **
960 REM ************************************
970 REM ************************************
980 :
990 REM **** PROGRAMA PRINCIPAL DE LLAMADA ****
1000 GOSUB 1600:REM LEER DATOS LABERINTO
1100 GOSUB 3700:REM IMPRIMIR LABERINTO
1200 GOSUB 5400:REM CONSTRUIR ARBOL
1210 GOSUB 7700:REM RECORRER ARBOL
1215 IF S(0)=9999 THEN PRINT "NO HAY SOLUCION": END
1220 GOSUB 8200:REM GUIAR VEHICULO
1400 END
1500 :
1600 REM **** LEER DATOS LABERINTO/INICIALIZAR ****
1700 READ SX,SY
1800 DIM MZ (SX,SY),TR(SX*SY,4),DR(4),RT(SX*SY),LN(SX*SY),CN(SX*SY)
1900 FOR Y=0 TO SX-1
2000 FOR X=0 TO SX-1
2100 READ A:MZ(X,Y)=A
2200 NEXT X,Y
2300 :
2400 READ XS,YS,XF,YF
2500 DATA 4,4
2600 DATA 1,0,0,0,0,0,1,0
2700 DATA 0,0,1,0,0,0,0,0
2800 DATA 1,1:REM COORDS PRINCIPIO
2900 DATA 2,3:REM COORDS FINAL
3000 :
3100 DR(1)=-SX:DR(2)=1:DR(3)=SX:DR(4)=-1
3110 ID(1)=3:ID(2)=4:ID(3)=1:ID(4)=2
3120 SR(0)=9999:REM INIC RUTA MAS CORTA
3130 RDD=56579:REGDAT=56577:POKE RDD,255
3140 AF=30:FF=45:REM FACTORES DE TIEMPO ANGULO Y ADELANTE
3200 FOR I=1 TO 25:CD$+CHR$(17):NEXT 1
3500 RETURN
3600 :
3700 REM **** IMPRIMIR LABERINTO ****
3800 PRINTCHR$(147):REM LIMPIAR PANTALLA
3900 FOR X=0 TO SX-1
4000 FOR Y=0 TO SY-1
4100 GOSUB 4900:REM POSICIONAR CURSOR
4200 PRINT CHR$(32+MZ(X,Y)*134)
4300 NEXT Y,X
4400 :
4500 X=XS:Y=YS:GOSUB4900:PRINT"P"
4600 X=XF:Y=YF:GOSUB4900:PRINT"LL"
4700 RETURN
4800 :
4900 REM **** POSICIONAR CURSOR EN X,Y ****
5000 PRINT CHR$(19);:PRINTTAB(X)LEFT$(CD$,Y);:RETURN
5300 :
5400 REM **** CONSTRUIR ARBOL ****
5500 :
5600 REM ** INICIALIZAR CON TERMINADORES ****
5700 FOR P=0 TO SX*SY-1:FORI=1T04:TR(P,I)=-1:NEXT I,P
6100 :
6110 REM ** CALCULAR PARTIDA Y LLEGADA ****
6120 X=XS:Y=YS:GOSUB9200:S=N
6130 X=XF:Y=YF:GOSUB9200:F=N
6140 :
6200 REM ** CONSTRUIR **
6300 LC=1:CC=0:REM INIC APUNTADORES PILA
6400 LN(LC)=S:REM PUNTO PARTIDA
6450 CN=LN(LC):REM TOMAR DE LA PILA NUDO EN CURSO
6500 DF=0
6600 FOR D=1 TO 4
6700 N=CN+DR(D):GOSUB 8900:REM CONVERTIR A X,Y
6800 IF(X<0 OR X>SX-1 OR Y<0 OR Y>SY-1)THEN 7300
6900 IF MZ(X,Y)=1 THEN 7300
7000 IF (D=2 AND N/SX=INT(N/SX))THEN7300
7100 IF (D=4 AND CN/SX=INT(CN/SX))THEN7300
7200 IF TR(N,ID(D))=CN THEN 7300
7210 TR(CN,D)=N:DF=1
7220 CC=CC+1:CN(CC)=N:REM PONER EN LA PILA
7300 NEXT D
7310 IF(DF=0 AND LC=1) THEN RETURN: REM NUDO TERMINAL
7320 :
7330 LC=LC-1:REM DECREMENTAR ULTIMO APUNT PILA
7340 IF LC>0 THEN 6450:REM PROXIMO NUDO
7345 :
7350 REM ** COPIAR PILA EN CURSO EN ULTIMA PILA **
7360 FOR I=1 TO CC:LN(I)=CN(I):NEXT I
7390 LC=CC:CC=0:GOTO6450:REM SIGUIENTE NUDO
7600 :
7700 REM **** RECORRER ARBOL ****
7720 C=0:RN=S:CN=RN:EF=0
7730 C=C+1:RT(C)=CN
7740 IF CN=F THEN GOSUB 8100:GOSUB8000:IF EF=0 THEN 7730
7745 IF EF=1 THEN RETURN
7750 DF=0
7760 FOR D=1 TO 4
7770 IF TR(CN,D)<>-1 THEN CN=TR(CN,D):DF=1:DR=D:D=4
7780 NEXT D
7790 IF DF=0 THEN GOSUB 8000
7800 IF EF=0 THEN 7730
7810 IF EF=1 THEN RETURN
7820 :
8000 REM *** RECOMENZAR POR LA RAIZ ****
8010 TR(RT(C-1),DR)=-1
8020 CN=RN:C=0
8030 IF(TR(CN,1)ANDTR(CN,2)ANDTR(CN,3)ANDTR(CN,4)=-1 THEN EF=1
8040 RETURN
8050 :
8100 REM **** GUARDAR MATRIZ ****
8110 IF C>SR(0) THEN RETURN:REM ES MAS CORTA LA RUTA NUEVA?
8120 SR(0)=C
8130 FOR I=1 TO C:SR(I)=RT(I):NEXT I:RETURN
8140 :
8200 REM **** GUIAR VEHICULO ****
8205 PD=1:REM SUPONER DIRECCION INICIAL ES NORTE
8210 FOR C=1 TO SR(0)-1
8220 DF=SR(C+1)-SR(C)
8222 :
8225 REM ** HALLAR DIRECCION REQUERIDA **
8230 FOR I=1 TO 4
8240 IF DF=DR(I) THEN D=I:I=4
8250 NEXT I
8260 :
8265 DR=D-PD:PD=D
8270 H=INT(4+DR/4):R=(4+DR)-4*H
8275 :
8277 REM ** EFECTUAR GIRO **
8280 FOR I=1 TO R
8290 POKE REGDAT=9:REM GIRO EN SENTIDO HORARIO
8300 T=TI
8310 IF (TI-T)<AF THEN 8310:REM ESPERAR
8320 POKE REGDAT=0:REM APAGAR
8330 NEXT I
8340 :
8350 REM *** ADELANTE ***
8360 POKE REGDAT=5
8370 T=TI
8380 IF (TI-T)<FF THEN 8380
8390 POKE REGDAT=0
8400 NEXT C
8410 :
8420 RETURN
8430 :
8900 REM **** CONVERTIR N A X,Y ****
9000 Y=INT(N/SX):X=N-SX*Y: RETURN
9200 :
9210 REM **** CONVERTIR X,Y A N ****
9220 N=Y*SX+X:RETURN
```

Para el BBC

Efectuar las siguientes modificaciones:

```
3130 RDD+&FE62:REGDAT=&FE60
8290 ?REGDAT=9
8300 TIME=0
8310 REPEAT UNTIL TIME>=AF
8320 ?REGDAT=0
8360 ?REGDAT=5
8370 TIME=0
8380 REPEAT UNTIL TIME>=FF
8390 ?REGDAT=0
```

El juego de la espada

Vamos a crear un juego de aventuras basado en texto. Lo haremos mediante un enfoque general para que usted pueda construir su propio juego

En este capítulo nos limitaremos a analizar los aspectos más generales de la programación de un juego de aventuras, y en el próximo consideraremos los detalles específicos para un juego en particular.

En todos los juegos de aventuras hay cinco actividades básicas que el jugador debe ser capaz de llevar a cabo: se necesita recoger objetos o abandonarlos, listar las cosas que uno lleva consigo, observar los alrededores y desplazarse por el juego de un cuarto a otro (o de escenario en escenario). De modo que son éstas las instrucciones básicas que vamos a programar antes que nada. Por razones de simplicidad, limitaremos la forma de las instrucciones a uno de dos tipos: o palabras individuales (como MIRAR) o bien pares compuestos por verbo y sustantivo (como ARROJAR ANILLO). El programa llevará dos listas: una denominada INVENTARIO, que incluirá todo lo que el jugador lleve consigo en cada momento, y la otra, llamada CONTENIDO, será una relación de los objetos existentes en el cuarto actual.

La primera instrucción que vamos a definir es INVENTARIO:

```
TO INV
  PRINT [UD. LLEVA:]
  IF EMPTY? :INVENTARIO THEN PRINT [NADA]
  ELSE PRINT :INVENTARIO
END
```

Observe que este procedimiento utiliza la forma completa de la sentencia IF: IF ‹condición› THEN ‹acción 1› ELSE ‹acción 2›. La instrucción para coger un objeto será COGER:

```
TO COGER :ITEM
  IF MEMBER? :ITEM :CONTENIDO
  THEN COGERLO :ITEM ELSE PRINT [NO
  PUEDO NO ESTA AQUI]
END
```

MEMBER? es una primitiva que verifica si un elemento pertenece a una lista. Para "coger" un artículo necesitamos hacer dos cosas: añadirlo al inventario y eliminarlo de la lista de contenido. Éstos son los procedimientos que realizan estas tareas:

```
TO COGERLO :ITEM
  AGREGAR.EN.INV :ITEM
  SACAR.DEL.CUARTO :ITEM
END
```

```
TO AGREGAR.EN.INV :ITEM
  MAKE"INVENTARIO SENTENCE :ITEM
  :INVENTARIO
END
```

```
TO SACAR.DEL.CUARTO :ITEM
  MAKE"CONTENIDO BORRAR :ITEM
  :CONTENIDO
END
```

El último de estos procedimientos implica borrar un elemento de una lista, tarea que fue uno de los ejercicios sugeridos en el capítulo anterior.

```
TO BORRAR :ITEM :LISTA
  IF:ITEM=FIRST :LISTA THEN OUTPUT BUTFIRST
  :LISTA
  OUTPUT SENTENCE FIRST :LISTA BORRAR :ITEM
  BUTFIRST :LISTA
END
```

La instrucción para dejar un objeto se implementa de forma similar:

```
TO DEJAR :ITEM
  IF MEMBER? :ITEM :INVENTARIO THEN
  DEJARLO :ITEM ELSE PRINT [NO PUEDES
  DEJARLO PORQUE NO LO TIENES!]
END
```

```
TO DEJARLO :ITEM
  SACAR.DEL.INV :ITEM
  AGREGAR.AL.CUARTO :ITEM
END
```

```
TO SACAR.DEL.INV :ITEM
  MAKE"INVENTARIO BORRAR :ITEM :INVENTARIO
END
```

```
TO AGREGAR.AL.CUARTO :ITEM
  MAKE"CONTENIDO FPUT :ITEM :CONTENIDO
END
```

Habiendo entrado todos los procedimientos que hemos dado hasta ahora, es el momento de verificar su funcionamiento. En primer lugar, debemos definir las dos variables globales (INVENTARIO y CONTENIDO) y después comprobarlas para las siguientes instrucciones:

```
MAKE"CONTENIDO [ESPADA LANZA ANTORCHA]
MAKE"INVENTARIO [FAROL]
COGER "ESPADA
DEJAR "FAROL
```

Ahora examine CONTENIDO e INVENTARIO utilizando estas sentencias:

```
PRINT :CONTENIDO
PRINT :INVENTARIO
```

y compruebe que estén correctas.

Observe que hemos empleado comillas antes de los nombres de los objetos tanto en la instrucción COGER como en DEJAR. El empleo de las comillas de esta forma ya es instintivo para el programador



de LOGO, pero es probable que le resulte muy confuso a un "aventurero" que no sepa nada acerca del lenguaje. Para posibilitar el empleo de la forma COGER ESPADA, que es más natural, debemos definir ESPADA de la siguiente manera:

```
TO ESPADA
  OP"ESPADA
END
```

Por supuesto, tendremos que hacer lo mismo con cada sustantivo que se utilice en nuestro juego de aventuras basado en texto.

La instrucción MIRAR imprimirá una descripción del cuarto en curso, una lista de lo que contenga y las posibles rutas de salida del cuarto. Para hacer esto necesitaremos otras dos listas: una primera de descripciones y una segunda de salidas. Con el objeto de poder incluir descripciones extensas que ocupen más de una línea de la pantalla, la lista de descripciones se define como una lista de listas. Por ejemplo:

```
MAKE"DESCRIPCION [[UD ESTA PARADO JUNTO
  A LA ENTRADA][DE UNA CUEVA]]
```

Para llevar un registro de cómo se unen los cuartos entre sí, cada cuarto tiene asignado un número. La lista de salidas es simplemente una lista de sublistas, cada una de ellas compuesta por una dirección y un número de cuarto.

Por tanto:

```
MAKE"LISTA.SALIDAS [[N 4][E 6]]
```

Ahora ya podemos definir MIRAR:

```
TO MIRAR
  PRINTL :DESCRIPCION
  PRINT "
  PRINT [UD VE:]
  IF EMPTY? :CONTENIDO THEN PRINT [NADA
  ESPECIAL]ELSE PRINT :CONTENIDO
  PRINT "
  PRINT [PUEDE IR:] IMPRIMIR. SALIDAS
  :LISTA.SALIDAS PRINT "
END
```

En este procedimiento se han empleado dos rutinas de impresión especiales para que la visualización resulte más fácil de leer. Se utiliza PRINTL para imprimir varias líneas de texto.

```
TO PRINTL :LISTA
  IF EMPTY? :LISTA THEN STOP
  PRINT FIRST :LISTA
  PRINTL BUTFIRST :LISTA
END
```

IMPRIMIR.SALIDAS imprime las salidas del cuarto sin imprimir los números de los cuartos.

```
TO IMPRIMIR.SALIDAS :LISTA
  IF EMPTY? :LISTA THEN PRINT "STOP
  MAKE "SALIDA FIRST :LISTA
  PRINT1 FIRST :SALIDA
  PRINT1 " ' '
  IMPRIMIR.SALIDAS BUTFIRST :LISTA
END
```

Podemos describir todo cuanto se sepa acerca de un cuarto del juego poniendo juntas las tres sublistas: la descripción, el contenido y las salidas. Por ejemplo:

```
MAKE "CUARTO.1 [[[UD ESTA PARADO JUNTO A LA
ENTRADA] [A UNA CUEVA]] [ESPADA] [[N 4][E 6]]]
```

Dado que CUARTO.1 se define de esta forma, podríamos dividirla en sus componentes individuales con el siguiente procedimiento:

```
TO ASIGNAR.VARIABLES
  MAKE "CUARTO THING "CUARTO.1
  MAKE "DESCRIPCION DESCRIPCION :CUARTO
  MAKE "CONTENIDO CONTENIDO :CUARTO
  MAKE "LISTA.SALIDAS LISTA.SALIDAS :CUARTO
END
```

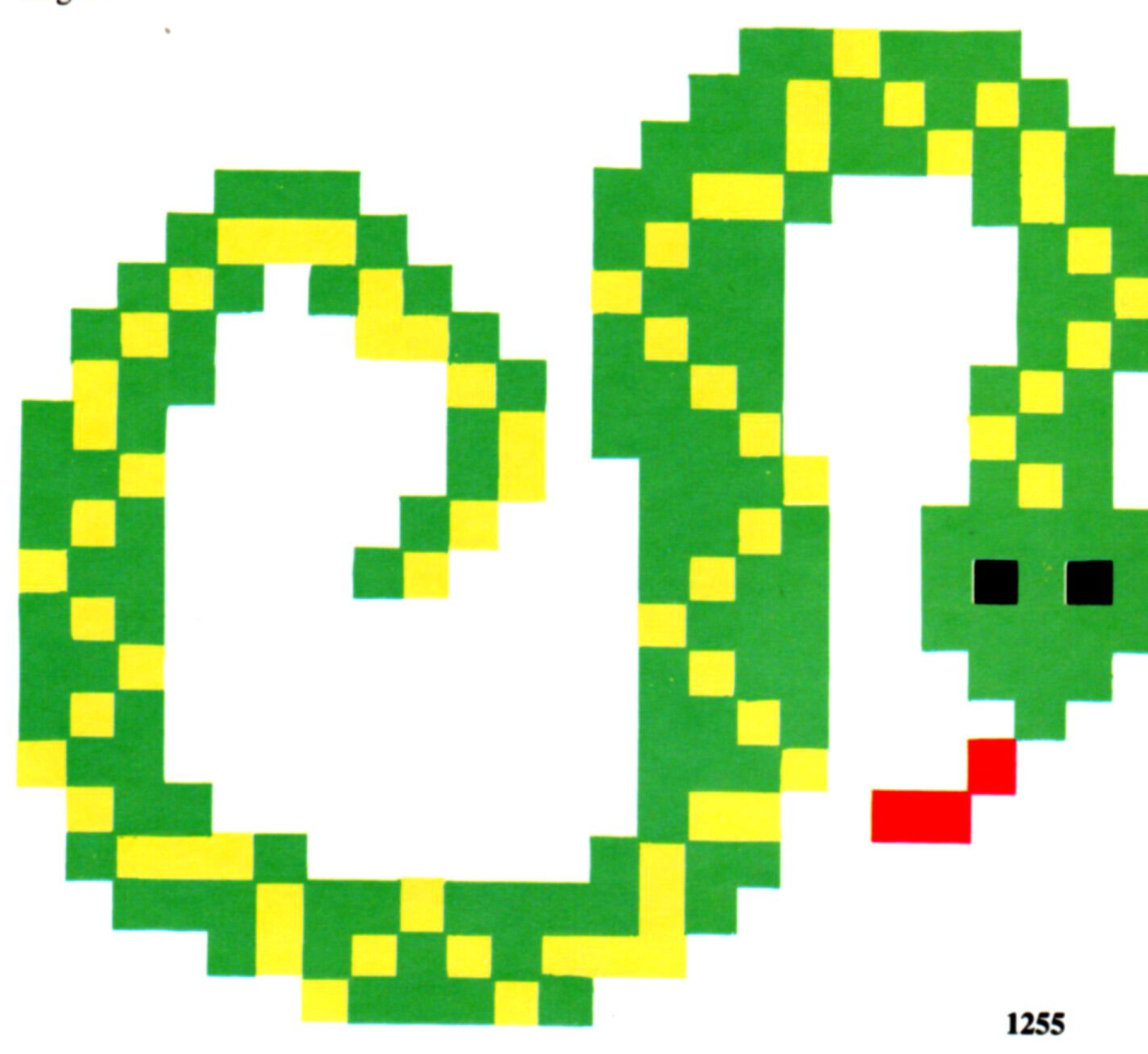
THING "CUARTO.1 es una alternativa de :CUARTO.1; significa "el contenido de la variable CUARTO.1". Luego explicaremos el motivo de la utilización de esta forma. Los subprocedimientos se definen de esta manera:

```
TO DESCRIPCION :CUARTO
  OUTPUT ITEM 1 :CUARTO
END

TO CONTENIDO :CUARTO
  OUTPUT ITEM 2 :CUARTO
END

TO LISTA.SALIDAS :CUARTO
  OUTPUT ITEM 3 :CUARTO
END
```

Tal como está, este procedimiento sólo funciona para CUARTO.1. Necesitamos ampliarlo de modo que se lo pueda utilizar con un carácter más general para cualquier cuarto. Esto lo hacemos empleando una variable global, AQUI, que contiene el número del cuarto en curso. Supongamos que ahora éste sea 2. La primitiva WORD del LOGO produce una palabra compuesta por una combinación de sus dos entradas (por tanto, WORD "CUARTO.:AQUI produciría CUARTO.1). Entonces le asignamos este nombre a la variable NOMBRE.CUARTO; en consecuencia, NOMBRE.CUARTO es CUARTO.2. Ahora podemos asignar



```
CUARTO como THING :NOMBRE.CUARTO.
  TO ASIGNAR.VARIABLES
   MAKE "NOMBRE.CUARTO WORD "CUARTO.:AQUI
   MAKE "CUARTO THING :NOMBRE.CUARTO
   MAKE "DESCRIPCION DESCRIPCION :CUARTO
   MAKE "CONTENIDO CONTENIDO :CUARTO
   MAKE "LISTA.SALIDAS LISTA.SALIDAS :CUARTO
END
```

Ahora ya está en condiciones de dibujar un mapa de los escenarios de su juego de aventuras y listar las descripciones de los mismos (con contenido y salidas). En el próximo capítulo acabaremos nuestro análisis general estudiando el movimiento entre escenarios y cómo se implementan los "peligros". Después comenzaremos a considerar un juego de aventuras completo como ejemplo de lo que se puede hacer.

Complementos al LOGO

Algunas versiones de LOGO MIT no poseen EMPTY?, ITEM, COUNT ni MEMBER? En el capítulo anterior ofrecimos definiciones para las tres primeras. La definición de MEMBER? es:

```
TO MEMBER? :ITEM :LISTA
   IF :LISTA = [] THEN OUTPUT "FALSE
   IF :ITEM = FIRST :LISTA THEN OUTPUT
"TRUE
   OUTPUT MEMBER? :ITEM BUTFIRST :LISTA
END
```

En todas las versiones LCSI utilice:

```
EMPTYP por EMPTY?
LISTP por LIST?
MEMBERP por MEMBER?
TYPE por PRINT1
```

También hay una primitiva, EQUALP, que comprueba si sus dos entradas son la misma. Utilícela para comparar listas y palabras en lugar de =. (El signo igual funciona para listas sólo en algunas versiones LCSI.) La sintaxis de IF completa en LOGO LCSI se demuestra mediante este ejemplo:

```
IF EMPTYP :CONTENIDO [PRINT [NADA
ESPECIAL]][PRINT :CONTENIDO]
```

Si la condición es verdadera se lleva a cabo la primera lista después de la condición, y si es falsa, se lleva a cabo la segunda.

Respuestas

1. Para imprimir en orden inverso:

```
TO PRINTR :LISTA
   IF EMPTY? :LISTA THEN PRINT "STOP
   PRINT1 LAST :LIST
   PRINT1" ' '
   PRINTR BUTLAST :LISTA
END
```

El siguiente procedimiento produce la lista en orden inverso, en lugar de imprimirla:

```
TO INVERTIR :LISTA
   IF EMPTY? :LISTA THEN OUTPUT []
   OUTPUT SENTENCE LAST :LISTA INVERTIR
   BUTLAST :LISTA
END
```

2. En el texto principal de este capítulo se incluye un procedimiento para borrar un elemento de una lista.

Poesía LOGO

En el capítulo anterior le planteamos el problema de mejorar las aptitudes del LOGO para escribir poesía. Un posible método consiste en crear un "modelo" de una estructura de frase (tal como "sustantivo, verbo, sustantivo") y elegir palabras de sublistas de estas partes de la frase. Por ejemplo, utilizando este modelo:

```
POEMA2 [SUSTANTIVO VERBO SUSTANTIVO]
```

podríamos tener:

```
TO POEMA2 :MODELO
   IF EMPTY? :MODELO PRINT "STOP
   POEMA2.1 FIRST :MODELO
   POEMA2 BUTFIRST :MODELO
END

TO POEMA2.1 :PAL
   IF :PAL="SUSTANTIVO (PRINT1" ' '
   GETRANDOM :SUSTANTIVO)
   IF :PAL="VERBO (PRINT2" ' '
   GETRANDOM :VERBO)
END
```

Este método sería muy incómodo si introdujéramos muchas más partes de la oración. Analicemos las variables más atentamente para ver si podemos mejorar algo. En primer lugar, entre:

```
MAKE "ROSA "DULCE
MAKE "OTRONOMBRE "ROSA
```

Ahora descubrirá que:

```
PRINT :ROSA imprime DULCE
PRINT :OTRONOMBRE imprime ROSA
PRINT THING :OTRONOMBRE imprime DULCE
```

THING nos da el valor asociado a un nombre. En el último caso, el nombre que sigue a THING es el valor de la variable OTRACOSA, o sea, ROSA. Utilizando esta idea, podemos reescribir nuestro procedimiento del poema:

```
TO POEMA2.1
   (PRINT1 " ' 'GETRANDOM THING :PAL)
END
```

La llamada a procedimiento POEMA2.1 "SUSTANTIVO le asigna a la variable PAL el valor SUSTANTIVO. Entonces el valor asociado con SUSTANTIVO es THING :PAL, la lista de sustantivos. Utilizando el modelo:

```
POEMA2 [ART SUSTANTIVO ADJ VERB ADV
PREP ART ADJ SUSTANTIVO]
```

junto con una adecuada selección de vocabulario, hemos obtenido la siguiente salida:

```
UN PLANETA VERDE GIRA RUIDOSAMENTE
BAJO UNA PARANOICA CUPULA
UNA NAVE RUINOSA VOLABA LENTAMENTE
HACIA EL ESPLENDIDO PLANETA
```



Errores no, gracias

Continuamos con la elaboración del programa depurador en lenguaje máquina iniciado en el capítulo anterior

Para el módulo de E/S hay que elaborar cuatro rutinas más: GETHX2, GETHX4, PUTHEX y PUTCR. Los dos primeros procesos sirven para tomar desde el teclado dígitos hexadecimales: GETHX2 se encarga de los números hexa de dos dígitos y GETHX4 de los de cuatro dígitos. Para diseñar estas rutinas, lo primero que hay que hacer es saber si vamos a solicitar la entrada siempre de dos o cuatro dígitos (lo cual es fácil de programar pero incómodo para el usuario) o bien aceptar un número de caracteres seguidos de un Return. El problema siguiente es decidir si se permitirá el carácter de retroceso para borrar los caracteres ya introducidos.

Vamos a adoptar el método más sencillo para la rutina GETHX4: sólo se dará entrada a cuatro dígitos y no se empleará el retroceso. El valor de 16 bits (que significa una dirección) puede retornarse en el registro D.

GETHX2 plantea más de un problema si consideramos las circunstancias en que será usado. Deberán entrarse cantidades de 8 bits para inspeccionar y cambiar la memoria (orden M), lo que significa acceder a una dirección. El contenido de ésta es visualizado, pudiendo a continuación el usuario pulsar Return (para pasar a la posición siguiente en secuencia) o bien un número hexadecimal de dos dígitos (para ser almacenado en esa posición) o incluso cualquier otro carácter (un punto, p. ej., para volver al nivel de órdenes). Podemos añadir los dos caracteres válidos para indicar el fin de una cadena de dígitos hexa. GETHX2 tendría, pues, que aceptar dos dígitos hexa, un Return o un punto. El valor de ocho bits se puede retornar en B y hay que emplear A para indicar en qué situación se está. Si lo que se ha introducido es un número de dos dígitos, A tendrá valor 0; si se trata de Return, tendrá valor 1, y si es un punto, valor −1. Tales valores no permiten comprobar el contenido de A sin compararlo con otro valor.

Supongamos de momento que para este módulo se han hecho las siguientes declaraciones:

```
HEXCHS     FCC'0123456789ABCDEF'
DOT        FCB '.'      (punto)
RETURN     FCB 13       (Return en código ASCII)
```

Podemos pasar 16 como la longitud de la cadena para GETHX4, donde sólo necesitamos dígitos hexa, y 18 para GETXH2, donde se necesitan también los otros dos caracteres.



Rutina GETHX2

Data:

- **Carácter-siguiente** es el ASCII contenido en A
- **Desplazamiento** a la tabla de Car-Válidos, está en B
- **Valor-hexa** se construye en B y es un valor de 8 bits
- **Flag** está en A y es 0,1 o bien −1

Proceso:

```
Tomar Carácter-Siguiente
IF carácter es un punto (Desplazamiento = 16)
  entonces
  Poner Flag a −1
ELSE si el carácter es un Return (Desplazamiento =
  17) entonces
  Poner Flag a 1
ELSE
  Guardar Desplazamiento temporalmente
  Tomar Carácter-Siguiente (en este punto sólo son
  válidos dígitos hexas)
  Construir Valor-Hexa
ENDIF
```

La forma final codificada de GETHX2 la encontrará el lector en la página 1259. Por otro lado, la codificación de la rutina GETHX4 se hace ahora más fácil con algunas piezas de la presente rutina. Haciendo HX4 un punto alternativo de entrada a la rutina GETHX2, es posible llamar a esa rutina y asegurarnos de que sólo se aceptan dígitos hexa válidos, siempre que carguemos B con un 16 antes de la llamada. De esta manera el proceso de toma de cuatro dígitos hexa se simplifica considerablemente.

Rutina GETHX4

Data:

- **Número-hexa**, es un valor de 16 bits que se retorna en D
- **Byte-Más-Significativo**, y
- **Byte-Menos-Significativo**, son dos valores de 8 bits que serán retornados en B

Proceso:

```
Tomar Byte-Más-Significativo
Guardar Byte-Más-Significativo temporalmente
Tomar Byte-Menos-Significativo
Construir Número-Hexa
```

Damos la codificación final de esta rutina después de la de GETHX2.

El diseño de las rutinas para la visualización de caracteres es mucho menos complicado. En la rutina PUTHEX, supondremos que el número de 8 bits que necesitamos se encuentra en B.

Rutina PUTHEX

Data:

- **Número**, es el valor de 8 bits que se encuentra en B
- **Desplazamiento**, es el desplazamiento de 4 bits colocado en HEXCHS

Proceso:

```
Extraer los 4 bits más significativos de Número para
```

emplearlos como Desplazamiento
Visualizar HEXCHS (Desplazamiento)
Extraer los 4 bits menos significativos de Número
Visualizar HEXCHS (Desplazamiento)

La rutina final necesaria para manejar las entradas y salidas es la subrutina PUTCR. Ésta es inmediata, y la codificación final se explica por sí misma. Una vez codificadas todas las rutinas que se necesitan, podemos ya diseñar el módulo entero de E/S.

Módulos de Entrada/Salida

Proceso:
Tomar Orden devolverá Desplazamiento en B, el cual puede servir de desplazamiento para una tabla de salto
Tomar Dirección deja la dirección de retorno en D
Tomar Valor deja el valor de retorno en B, flag en A
Visualizar Valor, pasado en B
Visualizar Dirección, pasado en D

La codificación final del módulo E/S se encuentra en la página siguiente. Ahora ya podemos volver al módulo de Puntos de Ruptura que iniciamos en el capítulo anterior (véase p. 1238). Ya dimos la codificación del segundo proceso de este módulo, que establece los puntos de ruptura. Pospusimos el problema de la codificación del primer proceso (insertar puntos de ruptura) porque presuponía la obtención de una dirección. Dado que esta tarea ya la hemos encarado en las rutinas que proporcionamos aquí, podemos seguir adelante ofreciendo la versión codificada del proceso, que incorpora una bifurcación a la subrutina GETADD.

Obsérvese que en el código, la instrucción INC NUMBP, PCR suma un 1 a Número-Puntos-Ruptura, que es el desplazamiento correcto en la tabla de puntos de ruptura. Sin embargo, la dirección se retorna en D, y esto destruirá el valor contenido en A,

La rutina GETHX2

```
GETHX2  LDB   # 18          Número-car-val
HX4     PSHS  X             Guarda el registro empleado
        LEAX  HEXCHS,PCR    Toma en X la dir. de Car-Válidos
        BSR   GETCH         Toma Carácter-Siguiente
IFOO    CMPB  # 16          Si Desplazamiento = 16
        LDA   # $FF         Poner el flag a -1 (en
                            complemento a dos)
        BRA   ENDFOO
        CMPB  # 17          Si Desplazamiento = 17
        LDA   # 1           Poner a 1 el flag
        BRA   ENDFOO
        LSLB                Desplaza B cuatro lugares a la
        LSLB                izquierda para formar el dígito
        LSLB                más significativo; B conserva el
        LSLB                desplazamiento en HEXCHS y por
                            ello el valor binario
        PSHS  B             Guarda B temporalmente
        LDB   # 16          Sólo son válidos dígitos hexa
        BSR   GETCH         Carácter-Siguiente
        ADDB  1,S+          Construye un número de 8 bits y
                            pierde B temporalmente
        PULS  X,PC
```

La rutina PUTCR

```
PUTCR   PSHS  A             Guarda A
        LDA   # 13          Return en código ASCII
        BSR   OUTCH         Lo visualiza
        PULS  A,PC
```

La rutina GETHX4

```
GETHX   LDB   # 16
        BSR   HX4           Toma Byte-Más-Significativo
        PSHS  B             Guarda Byte-Más-Significativo
                            temporalmente
        LDB   # 16
        BSR   HX4           Toma Byte-Menos-Significativo
                            en B
        PULS  A             Toma Byte-Más-Sign. para
                            devolverlo a A
        RTS                 El valor requerido está en D
```

Rutina Visualizar-Puntos-de-Ruptura

```
BPLABS  FCC   '1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16'
SPACE   FCB   32            Espacio en código ASCII
DISPBP  PSHS  A,B,X,Y
        LEAX  BPTAB,PCR     Dir. de Tabla-Puntos-Ruptura
        LEAY  BPLABS,PCR    Dir. de etiquetas
        CLRB                Pone en despl. 0 a Núm.-Punto-
                            Rupt.
WHIL01  CMBP  NUMBP,PCR     While Número-Punto-Ruptura <
                            = Número-Puntos-Ruptura
        BGT   ENDW01
        LDA   ,Y+           Visualiza etiqueta
        BSR   OUTCH
        LDA   ,Y+
        BSR   OUTCH
        LDA   SPACE,PCR     Visualiza un Espacio
        BSR   OUTCH
        PSHS  B             Guarda temporalmente B
        LDD   ,X++          Guarda Dirección
        BSR   DSPADD
        PULS  B             Restaura B
        BRA   WHIL01
ENDW01  PULS  A,B,X,Y       Restaura y retorna
```

puesto que el registro D comprende los registros A y B. Ésta es la razón por la que empleamos Y para conservar la dirección actual.

Habiendo codificado el primer proceso del módulo de Puntos de Ruptura, quedan aún tres procesos más por codificar. Dos de los cuales invierten los dos procesos que hasta ahora hemos codificado: Eliminar-Punto-Ruptura quitará de la tabla un punto de ruptura, mientras que Desactivar-Punto-Ruptura quita el Opcode SWI y devuelve el valor original. En el próximo capítulo estudiaremos estas dos rutinas. La tercera, que visualiza todos los puntos de ruptura, es la última que codificaremos.

Visualizar-Puntos-De-Ruptura

Data:

Número-Punto-Ruptura, es un contador de 8 bits para recorrer la tabla de Puntos de Ruptura; se encuentra en B

Punto-Ruptura-Actual, es la dirección a visualizar

Etiquetas-de-Puntos-Ruptura, son números (decimales) de dos dígitos para etiquetar las direcciones conforme son visualizadas

Espacio, es el carácter espacio que separa una etiqueta de una dirección

Proceso 3: Visualizar-Puntos-de-Ruptura

```
Poner Número-Punto-Ruptura a 1 (un
   desplazamiento actual de cero)
While Número-Punto-Ruptura < = Número-
   Puntos-Ruptura
      Visualizar Etiquetas-de-Puntos-Ruptura
         (Número-Punto-Ruptura)
      Visualizar Tabla-Puntos-Ruptura (Número-
         Punto-Ruptura)
      Incrementar Número-Punto-Ruptura
Endwhile
```

Fin de Proceso 3

La rutina PUTHEX

```
PUTHEX  PSHS  A,B,X          Guarda regs. empleados
        PSHS  B              Guarda B temporalmente
        LEAX  HEXCHS,PCR     Dir. de HEXCHS en X
        LSRB                 Desplazamiento de cuatro lugares
        LSRB                 para los bits más significativos
        LSRB
        LSRB
        LDA   B,X            Toma el carácter adecuado y lo
                             coloca en A
        BSR   OUTCH          Lo visualiza
        PULS  B              Toma de nuevo B
        ANDB  #%00001111     Enmascara los cuatro bits más
                             significativos
        LDA   B,X            Segundo carácter
        BSR   OUTCH
        PULS  A,B,X,PC       Restaura y retorna
```

Rutina Inserción-Puntos-Ruptura

```
BP01    PSHS  A,B,X,Y        Guarda regs. empleados
        LDX   BPTAB          Dirección de Tabla-Puntos-
                             Ruptura
        LDA   NUMBP,PCR
IF01    CMPA  MAXBP,PCR      Si Número-Puntos-Ruptura <
                             Max
        BGE   ENDF01
        INC   NUMBP,PCR      Sumar 1 a Número-De-Puntos
                             Ruptura
        LSLA                 Multiplica por dos el
                             Desplazamiento para la tabla de
                             16 bits
        LEAY  A,X
        BSR   GETADD         Toma la dirección
        STD   ,Y             Almacena la dirección en Tabla-
                             Puntos-Ruptura
ENDF01  PULS  A,B,X,Y        Restaura y retorna
```

El módulo Entrada/Salida

```
HEXCHS  FCC   '0 1 2 3 4 5 6 7 8 9 A B C D E F'
DOT     FCB
RETURN  FCB   13             Return en código ASCII
COMNDS  FCC   'BUDSGRMQ'
GETCOM  PSHS  A,X            Guarda los contenidos de A y X
        LEAX  COMNDS,PCR     Coloca la dirección de los
                             caracteres de la orden en X
        LDB   # 8            Número-Car-Vál
        BSR   GETCH          Toma Carácter
        PULS  A,X,PC         Return
GETADD  BSR   GETHX4
        BSR   PUTCR
        RTS
GETVAL  BSR   GETHX2
        BSR   PUTCR
        RTS
DSPVAL  BSR   PUTHEX
        BSR   PUTCR
        RTS
DSPADD  PSHS  B              Guarda B temporalmente
        TFR   A,B            Byte-Más-Significativo en B
        BSR   PUTHEX
        PULS  B              Recupera B
        BSR   PUTHEX
        PULS  B
        BSR   PUTHEX
        BSR   PUTCR
        RTS
```

Fuera de la ley

En este juego el usuario personifica a Mugsy, jefe de una banda de gángsters en la Norteamérica de los años treinta

El papel de Mugsy consiste en tomar diversas decisiones sobre el número de "clientes" de la banda en los que se puede confiar, cuánto dinero se puede gastar en municiones para la banda, y las sumas que se le deben abonar a la policía en concepto de soborno. Si no se gasta el dinero suficiente en armas, la banda será desarticulada. Además, si la policía no recibe suficiente dinero, allanará la caja de caudales de los gángsters y la incautará.

El principal personaje que aparece en la pantalla es Louey, el hombre que es la mano derecha de Mugsy. Al principio del juego Louey enuncia sucintamente las reglas. El jugador toma sus decisiones estratégicas en respuesta a las solicitudes de Louey y luego espera los resultados, lapso en el que se visualiza una de las dos pantallas animadas del juego. La primera pantalla muestra una taberna clandestina en la que un gángster le dispara a un rival. En la segunda se ve una calle donde un gángster, asomándose por la ventanilla de un coche, dispara una ráfaga de ametralladora.

En los muelles

En la taberna

Ian Mckinnell

El mundo del hampa
Éstas son dos de las escenas de Mugsy. La primera muestra parte de la fase de preguntas y respuestas del juego, en la cual Louey le está pasando a Mugsy un informe acerca de la cotización actual de los "clientes". La segunda escena pertenece a una de las secuencias animadas. Observe el contorno blanco alrededor del "hampón" de la escalera. Es un ejemplo del enmascaramiento de atributos de las celdas de caracteres

Tras la pausa vuelve a aparecer Louey con los resultados de las decisiones del año anterior. Siempre que se hayan apartado fondos suficientes para cubrir los diversos pagos que se deben efectuar, el año acabará con beneficios. Entonces comienza la siguiente vuelta, con una repetición de la rutina de preguntas y respuestas.

El marcador del jugador se ofrece al final del programa en forma de porcentaje. El marcador depende de cuánta "pasta" y cuántos clientes se hayan acumulado, así como durante cuánto tiempo haya logrado sobrevivir Mugsy.

Aparte de las pantallas de instrucciones que poseen una pequeña ventana en la parte inferior que refleja la cantidad actual de "hampones" y "pasta en la caja", etc., las pantallas se componen enteramente de imágenes gráficas. Éstas están dibujadas en un Spectrum utilizando el paquete para gráficos *Melbourne Draw*, de Melbourne House. Dado que una pantalla en alta resolución en el Spectrum ocuparía más de seis K de memoria, es evidente que se debe de haber aplicado alguna técnica de compresión de datos para introducir el código en la memoria libre disponible.

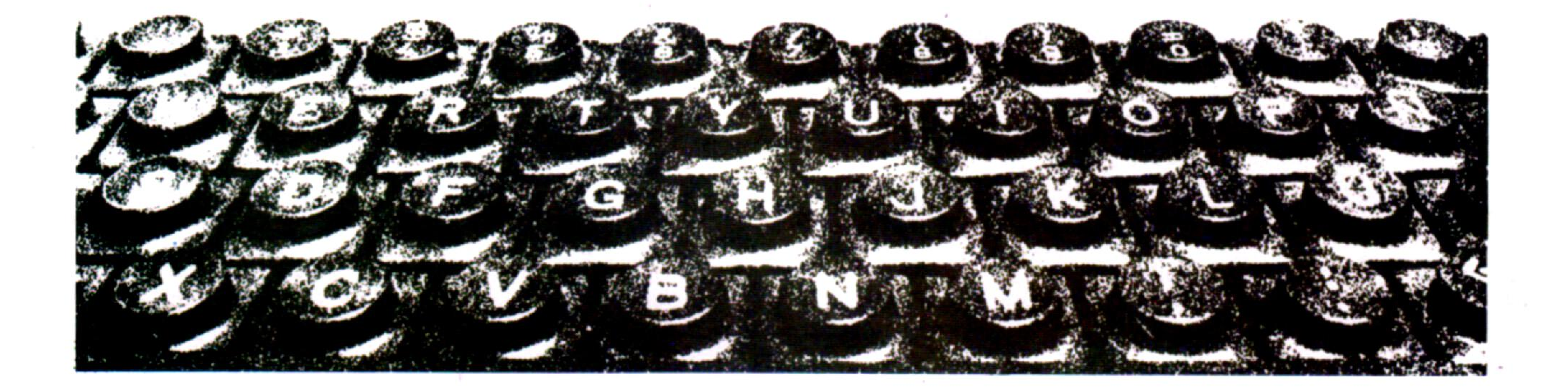
Un examen de los gráficos de *Mugsy* revela que los programadores han recurrido a numerosas técnicas para economizar espacio. En la mayoría de los casos, las imágenes están compuestas por una serie de líneas rectas y el color está aplicado con sumo cuidado. El empleo de instrucciones de *Melbourne Draw* como FILL y DRAW permite construir imágenes con una mínima cantidad de código; DRAW, por ejemplo, sólo requiere dos coordenadas para almacenar una línea recta, mientras que el trazado de un mapa de bits exigiría el trazado de toda una serie de puntos para conseguir el mismo resultado. La forma en que el Spectrum almacena la información relativa al color determina algunos de los métodos que se utilizan: en la escena animada de la calle, surge un problema cuando un coche avanza a lo largo de la calzada mientras se ve un rostro que observa desde una ventana. Puesto que el Spectrum no permite visualizar más de dos colores en el mismo cuadrado de carácter, se ha recurrido a una "máscara" para poder cambiar los colores rápidamente. Los atributos de color (FLASH, BRIGHT, INK y PAPER) se retienen en un único byte. Para producir una máscara de fondo se debe cambiar el atributo INK, retenido como los tres bits menos significativos del byte. El byte se opera primero mediante AND con 248 para establecer en cero el color INK, y éste se vuelve a operar luego mediante AND con el nuevo color INK para producir una máscara nueva. En realidad el rostro cambia de color para igualarse al color del coche, pero esto sucede tan rápidamente que se engaña al ojo y parece que el color del rostro no cambia en absoluto.

Los gráficos de *Mugsy* son ciertamente notables, pero el juego propiamente dicho aburre en seguida. La acción es repetitiva y el jugador aprende rápidamente a manejar los diversos factores necesarios para permanecer en el juego. No obstante, *Mugsy* representa un buen ejemplo de cómo introducir gráficos de alta resolución en un espacio limitado.

Mugsy: Para el Spectrum de 48 K
Editado por: Melbourne House, Church Yard, Tring, Herts
Autores: Phillip Mitchell, Greg Cull, Clive Barrett, Russell Comte
Palanca de mando: No se necesita
Formato: Cassette

mi computer

Editorial Delta, S.A.



10063
9 788485 822836